



The Uncertainty Principle



*The Bimonthly Collection of Creations
Presented by The Sophisticates Society*

Volume Orange Issue Four "Over the Horizon"

First e-book Edition

7 July 2012

\$0.00/Gratis

<http://theup.biz>

editor@theuncertaintyprinciple.biz

Table of Contents

I.	Page 3	Editor's Note
II.	Page 4	Editor's Rap
III.	Page 7	Paragogy: Synergizing individual and organizational learning
IV.	Page 18	Paragogia: Sinergia d'apprendimento individuale e organizzativa
V.	Page 34	ENG 099 Conversational American English Textbook 1.1st Edition
VI.	Page 61	Graded Lessons in English: An Elementary English Grammar
VII.	Page 355	The Adventures of Tom Sawyer
VIII.	Page 469	The Elements of Style
IX.	Page 517	Zen Style Programming
X.	Page 852	The Tao of Rork

Editor's Note

This issue is different from anything we've done before; typically our print editions are very zine-esque. This issue is more textbook, while still zine-y.

It starts with Joe Corneli and my first Paragogy Paper, plus a fantastic Italian translation by Fabrizio Terzi.

The heart of the work is an updated version of the ENG 099 Conversational American English Textbook. The work itself has been touched up and this work you're holding also contains all 3 complete books that my textbook references. Originally I just had samples, but I think with the other works included it's more comprehensive and better. Thank you Jan-Mark S. Wams¹ for making the PDF version of William Strunk's Elements of Style.

Closing out this issue are Nils H. Hom's "Zen Style Programming" and Peter St. Andre's novel "The Tao of Rork". Both men were kind enough to dedicate their books to the public domain.

On that note, this note, the following rap, all books and papers here are in the public domain. Please follow my lead here and re-use them in creative and novel ways! If you do, we here at the U.P. Would kindly appreciate you letting us know! Thanks.

Charles Jeffrey Danoff
The Uncertainty Principle, Editor
editor@theuncertaintyprinciple.biz

¹ <http://codeblab.com/elos/>

Editor's Rap

*World debut performance June 30th, 2012 at the U.P.'s Petite
Soirée № 7 in Albany Park, Chicago.*

Time to get down to business now.

Last Halloween there was this novella.

Its title was Keyi its 中文 4 Sucsexy.

Keyi its 中文 4 Sucsexy.

The title means "Keyi" you can.

Zhongwen is Mandarin for Chinese, Chinese for Mandarin.

Sucsexy is a portmanteau of successful and sexy.

The novella is set in-between a night out in Chicago, a dream and a hockey game.

Yeah!

So when I wrote it I didn't have a plan, or I didn't have a long outline.

I just woke-up every day and I wrote.

And whatever the muse or something came into my head I tried to put onto the page.

And, uh, so I wasn't trying to be a genre or anything. Just trying to take what came from wherever and put it on the page.

So, waht's it about you may be wondering?

If you've ever read Good Readers, Good Writers by Evengi Nabokov he says that uh ... anyway

The main thing is that the author doesn't know what it's about.
It's up to the readers to tell the author what it's about.

So if I said, like, 1 or 2 sentences on what it's about it would be limiting and wrong.

And, uh, the point is you have to tell me.

So here tonight, first off I want you to read it and tell me what it's about.

On a good note, it's Copyright licensed Public Domain, or dedicated to the Public Domain it has been given Unlicense in the back here
So, basically, you can do whatever you want with it.

Throw it anywhere, and re-mix it, you can make a movie out of it, make video game, make action figures of the characters, whatever you want.

You don't have to pay me anything, you don't have to tell me.
Anything.

I just want it to have idea sex. I want people to encounter it.

So, anyway, I published it last Halloween, and I've gotten a 100 copies printed so far

And, problem is, in one of the issues of The Uncertainty Principle last year I said more people to read it than people who've read the Bible.

So I'm a little behind.

So I'm here to ask for your attention and your dollars.

I can print it myself, I can license it ... I can do all this stuff myself, but in order to market it I need some help. Not my bag.

So, what I was thinking, for marketing I want a 1-page ad.

In the New York Times, London Times, Chicago Reader, Al Jazeera, Yomiuri Daily News a paper in Tokyo.

I want to do all that for the e-book.

People who donate to the e-book will get their name in the official, ultimate, clear edition forever to be enscribed.

So, yeah, so thank-you for bearing with me here, and, uh, I appreciate all of your attentions.

That's it.

On Friday, July 13th I will launch a Kickstarter campaign asking for support to market my novella. Thanks for reading this and please patronize the campaign while it's available.

[Log in](#) [Page Discussion](#) [View source](#) [History](#) [Zotero group](#)

[Main page](#)

[Recent changes](#)

[Issue Tracker](#)

[What links here](#)

[Related changes](#)

[Special pages](#)

[Printable version](#)

[Permanent link](#)

PARAGOGYPAPER1

Paragogy: Synergizing individual and organizational learning

by Joseph Corneli and Charles Jeffrey Danoff

published on Wikiversity (<http://en.wikiversity.org/wiki/User:Arieded/ParagogyPaper>) , January 2011

This paper describes a new theory of peer-to-peer learning and teaching that we call "paragogy". Paragogy's principles were developed by adapting the Knowles principles of andragogy to peer-based learning contexts. Paragogy addresses the challenge of peer-producing a useful and supportive context for self-directed learning.

The concept of paragogy can inform the design and application of learning analytics to enhance both individual and organization learning. In particular, we consider the role of learner profiles for goal-setting and self-monitoring, and the further role of analytics in designing enhanced peer tutoring systems.

INTRODUCTION

Jonathan Grudin identified several problems for computer supported collaborative work (CSCW), which apply *a fortiori* in computer supported collaborative learning (CSCL)^[1]. The current paper tackles similar problems, from a human and social perspective, in which both individual and organizational learning are front and center.

Grudin's thematic problems are: (1) The disparity between the people who do the work to create and support the application, and the people who get the benefit; (2) The breakdown of intuitive decision-making whenever intuition comes from a different context; and (3) the

Contents

- 1 Introduction
- 2 Paragogy: a theory of peer-based teaching and learning
- 3 Paragogical principles
 - 3.1 Paragogy compared with andragogy
 - 3.2 Paragogy and basho
- 4 Paragogy and Peer Production
 - 4.1 Context as a decentered center.
 - 4.2 Meta-learning as a font of knowledge.
 - 4.3 Peers are equals, but different.
 - 4.4 Learning is distributed and nonlinear.
 - 4.5 Realize the dream, then wake up!
- 5 Paragogy and Learning Analytics
 - 5.1 Context as a decentered center.
 - 5.2 Meta-learning as a font of knowledge.
 - 5.3 Peers are equals, but different.
 - 5.4 Learning is distributed and nonlinear.
 - 5.5 Realize the dream, then wake up!
- 6 Conclusion
 - 6.1 Next steps for the authors

ultimate difficulty of evaluating CSCW applications, precisely because they involve complex social dynamics.

6.2 Implementing paragogy
7 Acknowledgments
8 References

In the peer-based context, Problem 1 is somewhat mitigated, but by no means completely gone. Specializations tend to develop within every group. Power laws appear to distribute work between a core of dedicated users or contributors and a peripheral "long tail" of persons who are less involved.

We encounter Problem 2 as a direct side-effect of novelty. However, peer-based learning itself can more accurately be thought of as "new-old" (see Eisen ^[2]). Eisen's peer-based learning principles of voluntary involvement, trust, mutuality, authenticity, non-hierarchical status, and duration and intensity leading to closeness, are ways to describe fundamentally human situations (and quite nice-sounding ones at that). Perhaps these features are not as prevalent as they should be in our educational cultures; still the fact remains that it is not peer-based learning that is new, but many of the technologies that can support it (we count analytical methods and pedagogies among these).

We feel that Problem 3 is generally best handled by asking the people involved. If they are satisfied with their experiences, the systems involved are probably working reasonably well. If, on the other hand, they can identify some way the system could be improved, there may well be a chance to improve the system in a subsequent iteration. User feedback or even observation can thus comprise a "light" form of end-user development. That said, this approach merely transposes the problem of understanding social dynamics into a new, "technology-enhanced", version of the same problem. In any case, this will be a key problem for the nascent field of learning analytics.

In Section , we will describe our new theory of the social dynamics of peer-based education. In Section , we will develop the ideas further, relative to more general forms of peer production. Our views on how this new theory can inform the development of learning analytics are presented in Section . Finally, in Section , we describe some of our own planned work in this area, and suggest some other possible lines for future investigation.

PARAGOGY: A THEORY OF PEER-BASED TEACHING AND LEARNING

The theory of paragogy was developed in the context of two online courses that we ran at Peer 2 Peer University (P2PU) in Autumn of 2010. One of the courses was called "DIY Math", and it was "designed to build independent study and peer-support skills for mathematics learners at all levels." <http://p2pu.org/general/diy-math> The other course was called "Collaborative Lesson Planning", and it was built around the question "Can publishing and collaboratively building lesson plans online make them better?" <http://p2pu.org/general/collaborative-lesson-planning> The first course (which was facilitated by the first author of the current paper) was not a resounding success as a course, but we learned a lot from it anyway, especially in a rich discussion about how it could be improved that took place in the second course (which was facilitated by the second author).

The key outcome was an outline of an analytical framework that applies to peer-to-peer or peer-based teaching-and-learning-between-equals. The difficulties with DIY Math pointed to possible improvements at the organizational level, such as developing a P2PU-wide "social contract", or only running courses when sufficient commitments had been "anted up". In light of this, Corneli's post-mortem analysis of DIY Math suggested that the concept of pedagogy is not sufficient in the peer-based learning context; he then introduced the etymologically more appropriate term, paragogy. http://groups.google.com/group/diy-math/browse_frm/thread/5fcb82598445cd54 He subsequently five paragogical principles (Section), which were then improved and refined through a peer mentoring process in the Collaborative Lesson Planning course.

The fact that παραγωγή in is an existing word in Greek, meaning "generation" or "production", should not dissuade us from this new usage in English. Indeed, here we are precisely concerned with the activities that generate learning. And, vice versa, in the situated learning and communities of practice point of view, "learning was shown to be an inevitable aspect of all productive practices".^[3]

In any case, paragogy will be defined here in contradistinction to another neologism, andragogy, the teaching of adults, coined in ^[4] Cf. ^[5] ^[6]. We found Blondy's "Evaluation and Application of Andragogical Assumptions to the Adult Online Learning Environment" to be quite useful. ^[7] In succinct form, Knowles's five principles of andragogy are as follows: (1) that adult learners are self-directed; (2) that they bring a wealth of experience to the educational setting; (3) that they enter educational settings ready to learn; (4) that they are problem-centered in their learning; and (5) that they are best motivated by internal factors.

PARAGOGICAL PRINCIPLES

Each of these principles adjusts one of Knowles's five principles to the peer-based learning context, often by turning the original by 90°. This is not because we particularly disagree with Knowles about how to teach (see below), but because paragogy deals with a very different challenge, that of analyzing and co-creating the educational environment as a whole.

1. Context as a decentered center. "For learning design in a peer-to-peer context, understanding the learner's self-concept -- in particular, whether they see themselves as self-directed or not -- may be less important than understanding the concept of 'shared context in motion'." (See "Paragogy and basho", below.)

2. Meta-learning as a font of knowledge. "We all have a lot to learn about learning."

3. Peers are equals, but different. "The learner mustn't seek only to confirm what they already know, and must therefor confront and make sense of difference as part of the learning experience."

4. Learning is distributed and nonlinear. "Side-tracking is OK, but dissipation isn't likely to work. Part of paragogy is learning how to find one's way around a given social field."

5. Realize the dream, then wake up! "Paragogy is the art of fulfilling motivations when this is possible, and then going on to the next thing."

Paragogy compared with andragogy

Blondy points out both uses and challenges to each of Knowles principles of andragogy. For example, "Cheren stated that while learners may express a desire to be self-directed in their learning, most lack the required understanding of learning necessary to be self-directed and thus need guidance and encouragement in the learning process."

From our point of view, so much seems to depend on the way things are set up in the first place. For example, the most important initial condition in andragogy seems to be that an adult educator or facilitator is part of the picture. In a peer-based setting, that may not be the case: we can easily find examples of learning environments where there is no "teacher" in the "classroom"; where, for example, the task of facilitation is shared among all participants or even encoded in the learning materials or supportive technologies. Not that one way is more desirable than another: we simply mean to highlight the fact that the most basic features of a given learning environment will influence everything else.

In particular, it seems to us that a move to the more "horizontal" regime of paragogy can often occur within andragogy, e.g. when inviting participants to interact; and vice versa, a move to a more "vertical" regime of andragogy is possible within paragogy. For example, the second author fruitfully encouraged participation in his course via personal emails to those participants who had temporarily gone quiet.

In short, we agree with Blondy when she writes "Andragogy should be used as a starting point for approaching the adult online learning environment." We recommend paragogy as an additional starting point that sits on another dimension.

Paragogy and basho

The first paragogical principle stresses the importance of understanding the idea of shared context in motion. We will elaborate here.

The philosophical foundations of this notion, originally developed by Kitaro Nishida, and summarized in English by Masao Abe^[8], describe the way in which events and objects arise from their larger contexts. In other words, the idea of basho ("shared context in motion") can help us think about how a context constrains or supports different types of (inter-)actions, and also about how we (re-)shape the contexts we find ourselves in.

Nonaka and Toyama take this idea and apply it to knowledge creation. They suggest that knowledge is created as people interact over time in a shared context, in a process that can be broken up into repeated phases they call Socialisation, Externalisation, Combination, and Internalisation (SECI).^[9] In simple terms, any given phase can be understood in terms of "what I do", "what we do", "how we do it", and "what it's all about".

The first paragogical principle says that instead of focusing on how learners see themselves (e.g. as "self-directed" or "dependent" or something else), we should be asking how the learning context shapes what learners are actually able to do. Note that this includes looking at ways in which learners can contribute to reshaping the learning context.

Instead of simply saying "so-and-so lacks the required understanding of learning, so I need to help them", a paragogue would also look for contextual features of the learning environment that are "blocking" self-directed learning. These may include features that block the ability of learners to make adjustments to the environment on their own behalf, or which limit their ability to ask for help.

PARAGOGY AND PEER PRODUCTION

The links between paragogy and peer production illuminate both. As Phillip Schmidt writes: "Upon closer inspection of commons-based peer production communities, we find learning at their core".^[10] Conversely, in the conclusion to "Education and Mind in the Knowledge Age", Carl Bereiter writes:

Schools are places where knowledge creation can go on, but where it does not have to be market driven or competitive. [...] Knowledge creation in schools is the creation of knowledge by students for their own use. [...] To the extent that knowledge created in schools has value beyond the classroom where it was created, it enters into a barter economy.^[11]

Context as a decentered center.

The idea that internal motivation is in conflict with goal-directedness (from Tennant^[12], cited in Blondy) seems somewhat dubious if we consider the reciprocal effect of environment on character development described by Benkler and Nissenbaum.^[13]

Meta-learning as a font of knowledge.

Continuing this idea, gaining skills, employability, or a good reputation, seems to be a straightforward self-oriented way to enhance one's quality of life. But in fact, even these motivations come from somewhere. In a proper analytics of a learning or production landscape, we ought to ask: what learning? and why this learning?

Peers are equals, but different.

Benkler describes three necessary features for peer production: (1) the potential objects of peer production must be modular; (2) the modules must be small in size (noting that heterogeneous granularity will allow people with different levels of motivation to collaborate by contributing smaller or larger grained contributions); (3) the integration mechanism must run at a fairly low cost (either through automation or enforced social norms).

There are parallels in paragogy. The choice to work in a small closed group^[14] versus the choice to work as a group embedded within a larger commons^[15] has to do with the question: how much difference do you want to confront while engaging with the learning

process?

Learning is distributed and nonlinear.

The view of fluid social contexts advanced by Engestrom as a move beyond the traditional "communities of practice" view is quite compatible with the most famous peer production virtue, freedom (cf.), which is what allows people to function in a distributed and nonlinear fashion relative to a learning or production "ecosystem". Star and Griesemer^[16], on whom Wenger drew heavily as he was developing the idea of community of practice^[17], describe their view as "ecological". One key difference between Star/Wegner on the one hand and Engestrom on the other has to do with the nature of boundaries. In the community of practice view, boundary objects exist to effect translations or initiations. In Engestrom's view, attention is drawn to boundaries that remain in flux (via an ongoing process of co-configuration) or which are blurred (e.g. by a blurring of consumer and producer roles).

A closely related idea from Engestrom is that sociality revolves around concrete "shared objects", as opposed to e.g. abstract connections between people.^[18] Combining this with the idea of basho, we come to the at once intuitive and powerful idea of a context or environment as the largest shared object. An environment that is co-created by its inhabitants is likely to be a particularly meaningful and valued place.

Realize the dream, then wake up!

Blurred boundaries make it difficult to pinpoint a universally-applicable definition of "success". However, as Schmidt points out, measurable things like code commits can be used to make reasonably objective evaluations about participation in open source software projects , and we can expect to find other similar measurables related to modular contributions to other types of commons-based peer produced artifacts.^[19] It is may be in some ways more challenging to measure the (equally necessary) contributions to integration and coordination.

PARAGOGY AND LEARNING ANALYTICS

We now come to the paper's main application of paragogy, namely, to produce an outline that can give shape to the effectively infinite possibilities of learning analytics (henceforth, LA).

Context as a decentered center.

George Siemens defines learning analytics as "the use of intelligent data, learner-produced data, and analysis models to discover information and social connections, and to predict and advise on learning."^[20]

Measuring a student's progress in a given learning environment, whether it is centralized (Freshman to Senior) or decentralized (Padawan to Jedi), should suitably indicate the context of that student at each point. Progress may be defined relative to a context of the

activities of other participants in the environment. In a straightforward case, LA will be established and maintained relative to a changing collection of goals that are defined by an instructor or facilitator. LA will themselves be a nontrivial part of any learning context that employs them, suggesting that transparency about the way they are used will be an important factor to consider.

Continuing, student and instructor LA will increase institutional effectiveness, one example being the project Paul J. Williams is working on, "to supply student and organisational 'learner analytics' functionality to schools so that the decisions they make about the application of time and dollar resources amongst competing priorities can be better supported and justified. Institutions could see results for whether money invested in technology, teachers, facilities and more yields an improvement in learning for students, or not." <http://groups.google.com/group/learninganalytics/msg/fbd3385b8a86785f>

While applications of LA based on standardized tests is currently important, hopefully with more study the field will become more sophisticated and allow for a more holistic evaluation of learning than what is produced by standardized tests. In particular, this raises provocative question as to how best to measure school success.

Bereiter emphasizes developing a context that includes functional help for thinkers and learners, as opposed to applications of recieved wisdom about thinking or learning. He feels that thinking aloud research shows promise as a way to see just how people actually think (, p. 348). Paragogy suggests a broader view on thinking aloud: instead of traditional didactics, in a peer-based context, speech flows in a network, and thinking is done in an inherently social way.

Meta-learning as a font of knowledge.

Another definition given by EDUCAUSE's Next Generation learning initiative is "the use of data and models to predict student progress and performance, and the ability to act on that information". <http://nextgenlearning.com/the-challenges/learning-analytics>

In short, the meta-learning principle is the most obvious application of LA: the more effectively we can do LA, the more we learn about learning.

Peers are equals, but different.

Not only can LA be used to measure an individual student's successes, failures, and hours invested, they can be applied in relation to data about peers, including peer-facilitators or teachers (e.g. in connection with suggestions or critiques). LA could be used to pair up weaker students with more advanced ones, or to help learners with overlapping interests find each other in the crowd. Threshold values could be set to indicate when a student might be allowed or asked to move from a mentored to mentoring role. Measurements can also be made of how well students work with their peers, or how much they have individually contributed to the learning environment.

Data on how different learners appear to learn best could be combined with information on how certain tutors work to find the best pairing. Various other sorts of recommendations are part of the subject of a significant body of ongoing research.^[21]

Learning is distributed and nonlinear.

LA, especially attention metadata, can measure how much a student stays on topic, and give feedback on how these attentional investments pay off in the long-run. When logged into the class wiki, do they work for an entire hour on one page, or do they move around? Topics a student touches on but later abandons should be kept track of (for instance, because they may be useful later).

Students can show off their learning on things they may not have a degree in. For example, someone who majored in English in college who wants a career change can show an engineering firm school they independently completed "90 of the work towards a Journalism BA" to prove they have the skills and motivation for an entry-level job in public relations.

To get to point where a system can give feedback of this nature, goals need to be specified and agreed upon. Long-term goals would probably be easy enough, e.g. "I want to learn Japanese." It is harder to break a task into steps, and the first step is often the hardest. Corneli suggested to look for "the simplest step (that you can actually do) that gets you toward your goal." <http://www.p2pu.org/general/node/5571/document/10225>

Students can then share that step however small, and once achieved, can chose another one along the way. These patterns can be studied to find LA that will show a learner their percentage towards e.g. fluency in Japanese.

Realize the dream, then wake up!

We feel this is the key to combining LA and paragogy: a student should explicitly spell-out their motivations/goals and then keep track of their progress towards reaching them. LA will help students have a clear way to know how close they are to realizing their dreams, and to have a way to showcase their achievements to the rest of the world. In cases of trouble, LA should help identify how changes in behavior can help.

To think highlight here one possible large-scale application, we can imagine creating paragogical accreditation standards for learners, along the lines of those used for businesses by the Better Business Bureau. <http://www.bbb.org/us/bbb-accreditation-standards/> This could come from a system to that would keep track of the kinds of courses people might like to take; and furthermore, courses could require people to ante up a certain level of commitment before the course would run. The degree to which people follow through on their commitments over time would determine their credibility rating.

CONCLUSION

We explored connections between paragogy and peer production, and paragogy and learning analytics, and showed how paragogy can intertwine with these to open new avenues for productivity, learning, and evaluation.

Next steps for the authors

We both plan to try running courses on Peer 2 Peer University again when the next round

begins in January 2011. We will write syllabi that encourage paragogical activity while generating LA for evaluation. Another avenue we are exploring is creating our own learner profiles, as suggested by Siemens . http://en.wikiversity.org/wiki/User:Aided/Learner_Profile http://en.wikiversity.org/wiki/User:Charles_Jeffrey_Danoff/Learner_Profile Building on his framework, we will endeavor to maintain an outline our learning goals, steps to complete them, and criteria for evaluation. We will do what we can to encourage P2PU to support learner profiles across the board.

The ideas from Section will be further developed in an extensive case study by the first author on commons based peer-production in mathematics.^[22]

Implementing paragogy

We encourage the research community to test our ideas in practice of various forms. Some ideas for paragogical design include:

- Establish a group consensus for expectations/goals/social contract of the course and how each of them should be evaluated at its conclusion.
- Have learners designate learning goals that they then commit to stick with.
- Formalize a process for assisting peers (e.g. responding to questions, giving feedback on publicly posted work).
- Develop explicit pathways for learner feedback to translate into changes to the learning environment.

ACKNOWLEDGMENTS

Joseph Corneli's work is partially funded through the ROLE Integrated Project, part of the Seventh Framework Programme for Research and Technological Development (FP7) of the European Union in Information and Communication Technologies.

We wish to thank Marisa Ponti, Alexander Mikroyannidis, Raymond Puzio, and Thomas Lynch for their help with ideas and references. Thanks also to everyone at P2PU. Finally, thanks to the organizers of Wikimania 2010, where the authors first met and exchanged the early inklings of the ideas developed here.

REFERENCES

1. ↑ Jonathan Grudin: Why CSCW applications fail: problems in the design and evaluation of organizational interfaces. Proceedings of the 1988 ACM conference on Computer-supported cooperative work, pp. 85--93 (1988).
2. ↑ Mary-Jane Eisen: Peer-Based Learning: A New-Old Alternative to Professional Development. Adult Learning, 12(1) pp. 9--10 (2001).
3. ↑ Y. Engestrom: From communities of practice to mycorrhizae, in H. Hughes, N. Jewson, L. Unwin (Eds.), Communities of practice: Critical perspectives. London: Routledge (2007). (<http://www.open.ac.uk/cetl-workspace/cetlcontent/documents/476902341f33c.pdf>)

4. ↑ Kapp, Alexander: Platon's Erziehungslehre, als Padagogik fur die Einzelnen und als Staatspadagogik. Minden und Leipzig (1833).
5. ↑ M. S. Knowles: Andragogy, not pedagogy. *Adult Leadership*, 16(10), pp. 350--352 (1968).
6. ↑ M. S. Knowles: The modern practice of adult education: From pedagogy to andragogy. Chicago: Follett. (1980)
7. ↑ Laurie C. Blondy: Evaluation and Application of Andragogical Assumptions to the Adult Online Learning Environment, in *Journal of Interactive Online Learning*, 6(2) (2007) (<http://www.ncolr.org/jiol/issues/getfile.cfm?vollID=6&IssueID=20&ArticleID=104>)
8. ↑ Masao Abe: Nishida's Philosophy of 'Place'. *International Philosophical Quarterly* 28 (4), pp. 355--371 (1988).
9. ↑ I. Nonaka, R. Toyama, N. Konno: SECI, Ba and Leadership: a Unified Model of Dynamic Knowledge Creation. *Long Range Planning*, 33(1), pp. 5--34 (2000).
10. ↑ J. P. Schmidt: Commons-based peer production and education. Short essay for the Free Culture Research Workshop, Harvard University (23 October 2009). (http://cyber.law.harvard.edu/fcrw/sites/fcrw/images/Schmidt_Education_FreeCulture_25Oct2009.pdf)
11. ↑ Carl Bereiter: Education and Mind in the Knowledge Age. Lawrence Erlbaum Associates (2002)
12. ↑ M. Tennant: Psychology and adult learning. London: Routledge (1997)
13. ↑ Y. Benkler and H. Nissenbaum: Commons-based peer production and virtue. *Journal of Political Philosophy*, 14(4), pp. 394--419 (2006).
14. ↑ Robert Milson and Aaron Krowne: Adapting CBPP platforms for instructional use. In Martin Halbert, editor, *Symposium on Free Culture and the Digital Library*, pp. 255--272 (2005). (<http://arxiv.org/pdf/cs/0507028>)
15. ↑ Piotr Konieczny: Wikis and Wikipedia as a teaching tool. *International Journal of Instructional Technology and Distance Learning*, 4(1) (2007). (<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.113.7307&rep=rep1&type=pdf>)
16. ↑ Susan Leigh Star and James R. Griesemer: Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39. *Social Studies of Science*, 19(3) (1989), pp. 387-420.
17. ↑ E. Wenger: Toward a theory of cultural transparency: Elements of a social discourse of the visible and the invisible. PhD Dissertation in Information and Computer Science, University of California, Irvine (1990). (<http://www.ewenger.com/pub/pubEWdissertation.doc>)
18. ↑ Jyri Engestrom: Why some social network services work and others don't -- Or: the case for object-centered sociality, published online at <http://www.zengestrom.com/blog/2005/04/why-some-social-network-services-work-and-others-dont-or-the-case-for-object-centered-sociality.html>.
19. ↑ Y. Benkler: Common wisdom: Peer production of educational materials. Center for Open and Sustainable Learning at Utah State University, (2005). (http://www.benkler.org/Common_Wisdom.pdf)
20. ↑ George Siemens: What are Learning Analytics? published online at <http://www.elearnspace.org/blog/2010/08/25/what-are-learning-analytics/>.
21. ↑ Uwe Kirschenmann, Maren Scheffel, Martin Friedrich, Katja Niemann and Martin

Wolpers: Demands of Modern PLEs and the ROLE Approach, in Sustaining TEL: From Innovation to Learning and Practice, Springer LNCS, Volume 6383/2010, pp. 167--182 (2010).

22. ↑ Joseph Corneli: Crowdsourcing a Personalized Learning Environment for Mathematics, Knowledge Media Institute, Technical Report, (<http://metameso.org/~joe/docs/probation-report-final.pdf>)

Retrieved from "<http://paragogy.net/ParagogyPaper1>"

This page was last modified on 18 January 2012, at 17:52.
Content is available under CC0 1.0 Universal.



KNOWLEDGE MEDIA INSTITUTE, THE
OPEN UNIVERSITY, UK

Paragogia

Free Technology Guild for Beginner Level -1- V1.1 003:ITA
Traduzione e revisione Italiana a cura di: Fabrizio Terzi: siar@member.fsf.org

Sinergia d'apprendimento individuale e organizzativa

Author:

Mr. Joseph CORNELI
j.a.corneli@open.ac.uk

Author:

Mr. Charles J.DANOFF
contact@mr.danoff.org

22 maggio 2012

Sommario

QUESTO ARTICOLO, descrive la nuova teoria del peer-to-peer learning (condivisione di studio) applicata alla metodica educativa che noi chiamiamo Paragogia. I suoi principi, sono stati ipotizzati interpretando le dinamiche d'apprendimento infantili formulate da Malcolm S. Knowles associate ai contesti di studio e formazione (file-based) per adulti ben descritti dall'andragogia. La paragogia, affronta la sfida dell'apprendimento autodidattico in un contesto di peer-producing utile al supporto di studio sia individuale sia organizzato in gruppi di lavoro. In questo contesto, vengono evidenziati il ruolo e il profilo dello studente, la definizione dei suoi obiettivi (goal-setting) e il suo auto monitoraggio dei risultati ottenuti (self-monitoring). Questa discussione ad ogni modo, offrirà una ulteriore analisi nella progettazione e sviluppo partecipato di un più ampio sistema di affiancamento e tutoraggio didattico (teorico-pratico).

Key words: PEER-TO-PEER, PEDAGOGIA, ORGANIZZAZIONE, ANDRAGOGIA, APPRENDIMENTO ANALITICO.

Distribuito in Creative Commons CC0 1.0 universe.

1 Introduzione

Jonathan Grudin, identifica diverse problematiche nel lavoro condiviso supportato da sistemi informatizzati CSCW, che a maggior ragione, sono presenti durante il supporto collaborativo nelle fasi di apprendimento CSCL. Il presente articolo, presenta e affronta problemi analoghi in una prospettiva sociale in cui l'apprendimento individuale e organizzativo, ne sono la sua più viva componente attiva.

I principali problemi evidenziati da Grudin sono:

1. La disparità tra le persone che svolgono il lavoro partecipato per creare o supportare una iniziativa e le persone che ne ricevono il beneficio derivante;
2. La ripartizione delle processo decisionale quando l'intuizione proviene da un contesto differente;
3. La difficoltà nella valutazione delle applicazioni CSCW per la natura complessa delle dinamiche sociali al suo interno.

In un contesto basato sulla condivisione della conoscenza, il primo problema viene mitigato ma non del tutto superato nei contesti educativi. Particolari specializzazioni infatti, tendono a svilupparsi naturalmente all'interno di ogni gruppo seguendo una distribuzione paretiana (power-laws) che distribuisce in questo modo il lavoro tra un nucleo di utilizzatori e collaboratori dedicati, e una periferica lunga coda di persone meno coinvolte attivamente ma potenzialmente interessate. Come effetto collaterale di questa differenziazione incontriamo il problema (2) ben spiegato da Eisen [2]. I principi di apprendimento (peer-based) di Eisen comprendono:

- La partecipazione volontaria e la fiducia.
- Reciprocità nei rapporti.
- Autenticità.
- Una struttura gerarchia indipendente dallo status di provenienza.
- La durata e vicinanza delle relazioni tra i suoi partecipanti.

Tutte queste sono caratteristiche nobili che dovrebbero ulteriormente essere sostenute e sviluppate nei futuri modelli pedagogici e sistemi educativi. (Consideriamo che siamo solo agli inizi di questo cambiamento culturale). Il terzo

Distribuito in Creative Commons CC0 1.0 universe.

1 INTRODUZIONE

problema, è generalmente meglio gestito dalle stesse persone direttamente coinvolte. Se si è soddisfatti nella esperienza di partecipazione attiva, i sistemi utilizzati funzioneranno generalmente meglio e riceveranno sempre nuova energia dai suoi membri. Inoltre, qualora fossero identificate soluzioni migliorative, queste potranno essere meglio discusse in una successiva iterazione fra i suoi partecipanti. Il feedback degli utenti e una osservazione esterna libera, comprende una forma chiara di sviluppo e perfezionamento finale. Ciò detto, questo tipo di approccio si limita a trasporre il problema della comprensione delle dinamiche sociali ad un nuova e tecnologicamente più avanzata interpretazione del problema. Ad ogni modo, questo sarà un punto di discussione ed analisi nel campo di ricerca nascente. Nella sezione [2], descriveremo la nostra nuova teoria sulle dinamiche sociali di educazione peer-to-peer or peer-based. Nella sezione [3], si svilupperanno le nuove idee relative a forme più generale di produzione condivisa. I nostri pareri su come questa nuova teoria sia in grado di migliorare lo sviluppo d'analisi e apprendimento sono presentati nella sezione [4]. Infine, nella sezione [5], sono descritte le nostre linee guida di lavoro e alcuni possibili campi d'indagine futuri.

Distribuito in Creative Commons CC0 1.0 universe.

2 Paragogia: La nuova teoria per l'apprendimento e l'insegnamento (peer-based)

LA teoria della paragogia è stata ideata durante lo svolgimento di due corsi svolti on-line dalla **-Peer 2 Peer University -(P2PU)-** nell'autunno del 2010. Il primo corso si chiamava **DIY Math** (matematica fai da te), e fu ideato al fine di sviluppare e migliorare il supporto e le competenze necessarie agli studenti di matematica a tutti i livelli. Il secondo corso, fu chiamato **Collaborative Lesson Planning** e venne concepito nel tentativo di rispondere alla domanda «*Condividendo lo sviluppo e la progettazione on-line è possibile ideare piani di lezione e sistemi d'apprendimento migliori ?*». Il primo corso, (che è stato promosso dal primo autore di questo documento) in verità non ha avuto un successo clamoroso, ma ha permesso di raffinare e arricchire gli interventi dei partecipanti al corso successivo (facilitato dal secondo autore del documento). Un risultato importante già ottenuto, è stato l'individuazione di uno schema analitico applicabile al peer-to-peer durante i processi di insegnamento e apprendimento in contesti file-based. Durante il corso DIY Math, sono stati evidenziati possibili miglioramenti sul piano organizzativo riguardanti lo sviluppo di un più ampio accordo **-P2PU-** (accordo sociale di condivisione) circa le modalità di preparazione ai corsi futuri. Alla luce di questo, l'analisi offerta dal **Dr.Joe Corneli**, ha suggerito che il concetto di pedagogia risultava essere non più sufficiente in questi nuovi contesti; In questa occasione venne introdotto per la prima volta il termine Paragogia. Successivamente, sono stati introdotti i cinque principi fondanti migliorati e raffinati durante le lezioni di tutoraggio nel corso Collaborative Lesson Planning. Ad ogni modo, il concetto di paragogia verrà qui definito in contrapposizione ad un neologismo simile, l'andragogia che descrive un modello incentrato sui bisogni e gli interessi di apprendimento degli adulti, evolvendo e applicando le idee di Blondy ai processi d'apprendimento on-line.

I sei principi dell'andragogia:

fonte wikipedia:

1. **Il bisogno di conoscere:** gli adulti sentono l'esigenza di sapere perché occorra apprendere qualcosa. Tough (1979) ha scoperto che quando gli adulti iniziano ad apprendere qualcosa per conto loro investono una considerevole energia nell'esaminare i vantaggi che trarranno dall'apprendimento. Il primo compito del facilitatore dell'apprendimento è aiutare i discenti in questo risveglio di consapevolezza (Freire): egli può

Distribuito in Creative Commons CC0 1.0 universe.

2 PARAGOGIA: LA NUOVA TEORIA PER L'APPRENDIMENTO E
L'INSEGNAMENTO (PEER-BASED)

addurre come minimo degli argomenti sul valore dell'apprendimento nel migliorare l'efficienza della performance dei discenti o della loro qualità di vita.

2. **Il concetto di sé del discente:** man mano che una persona matura e diventa adulta, il concetto di sé passa da un senso di totale dipendenza ad un senso di crescente indipendenza ed autonomia. L'adulto deve sentire che il proprio concetto di sé viene rispettato dall'educatore e quindi deve essere collocato in una situazione di autonomia (contrapposto a una situazione di dipendenza).
3. **Il ruolo dell'esperienza:** la maggiore esperienza degli adulti assicura maggiore ricchezza e possibilità d'utilizzo di risorse interne. Qualsiasi gruppo di adulti sarà più eterogeneo – in termini di background, stile di apprendimento, motivazioni, bisogni, interessi e obiettivi – di quanto non accada in gruppi di giovani. Da qui deriva il grande accento posto nella formazione degli adulti sull'individualizzazione delle strategie d'insegnamento e di apprendimento, sulle tecniche esperienziali piuttosto che trasmissive e sulle attività di aiuto tra pari. La maggiore esperienza può avere anche tratti negativi, nel senso di una maggiore rigidità negli abiti mentali, delle prevenzioni, delle presupposizioni e nella chiusura rispetto a idee nuove e diverse modalità di approccio. Un'altra ragione che sottolinea l'importanza dell'esperienza è che, mentre per i bambini l'esperienza è qualcosa che capita loro, per gli adulti essa rappresenta chi sono. Essi cioè tendono a derivare la loro identità personale dalle loro esperienze.
4. **La disponibilità ad apprendere:** quanto viene insegnato deve migliorare le competenze e deve essere applicabile in modo efficace alla vita quotidiana.
5. **L'orientamento verso l'apprendimento:** non deve essere centrato sulle materie ma sulla vita reale. Gli adulti infatti apprendono nuove conoscenze, capacità di comprensione, abilità e atteggiamenti molto più efficacemente quando sono presentati in questo contesto. Questo punto ha un'importanza cruciale nelle modalità di esposizione dell'insegnante, degli obiettivi e nei contenuti definiti e nella progettazione più generale dell'intervento formativo.
6. **La motivazione:** nel caso degli adulti le motivazioni interne sono in genere più forti delle pressioni esterne. Tough (1979) ha scoperto che tutti gli adulti sono motivati a continuare a crescere e a evolversi, ma

Distribuito in Creative Commons CC0 1.0 universe.

2 PARAGOGIA: LA NUOVA TEORIA PER L'APPRENDIMENTO E
L'INSEGNAMENTO (PEER-BASED)

che questa motivazione spesso viene inibita da barriere quali un concetto negativo di sé come studente, l'inaccessibilità di opportunità o risorse, la mancanza di tempo e programmi che violano i principi dell'apprendimento degli adulti. In questo gioca anche un ruolo fondamentale la promozione dell'autodeterminazione, soddisfacendo i bisogni psicologici innati di competenza, autonomia e relazione. La competenza consiste nel sentirsi capaci di agire sull'ambiente sperimentando sensazioni di controllo personale. L'autonomia si riferisce alla possibilità di decidere personalmente cosa fare e come. Il bisogno di relazione riguarda la necessità di mantenere e costituire legami in ambito sociale.

Distribuito in Creative Commons CC0 1.0 universe.

3 Principi Paragogici

Ognuno dei principi di Knowles, è stato adattato ai contesti d'apprendimento peer-base della paragogia spesso capovolgendone il significato. Questo non perché siamo in disaccordo con Knowles (vedi Sezione 2.2), ma piuttosto perché la sfida della paragogia analizza l'ambiente e la co-creazione educativa nel suo complesso in senso più ampio.

1. **Contesto come centro decentrato:** Per l'apprendimento e la progettazione in una rete peer-to-peer, la comprensione dello studente al self-concept, in particolare (come autodidatta o con affiancamento), risulta meno importante della comprensione del concetto di *contesto condiviso in movimento*. (Si veda la Sezione 2.3.)
2. **Meta-learning come fonte di conoscenza:** Tutti hanno la possibilità di continuare a migliorarsi e apprendere cose nuove utili al progresso.
3. **I collaboratori sono uguali e differentemente unici:** Gli studenti non devono ricercare solo per confermare ciò che già sanno, ma devono affrontare e dare un senso alla differenza dei punti di vista come parte dell'esperienza di apprendimento.
4. **L'apprendimento è distribuito e non lineare:** il Side-tracking può funzionare, ma la dissipazione non è in grado di lavorare al meglio. Parte della paragogia consiste nell'imparare e trovare la propria strada attorno ad un dato campo sociale.
5. **Realizzare il proprio perché e poi viverlo!** La Paragogia è l'arte di comprendere le proprie motivazioni, individuare i propri obbiettivi, e poi saper procedere passo dopo passo fino al traguardo.

Distribuito in Creative Commons CC0 1.0 universe.

4 Paragogia e Andragogia a confronto

L AURIE Cheren Blondy, [7] evidenzia le implicazioni pratiche e le nuove sfide che l'andragogia di Knowles si trova ad affrontare. Ma Come? Blondy, ha dichiarato che; mentre gli studenti possono esprimere il desiderio di apprendere in modo autodidattico, nella maggior parte dei casi non adottano una metodologia di studio efficace e necessitano quindi di una guida a supporto e un forte incoraggiamento e sostegno iniziale. Dal nostro punto di vista, molto sembra dipendere dalla impostazione metodica iniziale. A conferma, la condizione più importante di partenza dell'andragogia sembra essere il coinvolgimento di un educatore esperto o facilitatore fin dall'inizio delle fasi di apprendimento. In una rete peer-based al contrario, questa condizione d'impostazione può essere differente: non è raro trovare oggi, ambienti di apprendimento dove non sia presente un insegnante di classe, dove il ruolo di facilitatore, viene spesso suddiviso tra i suoi stessi partecipanti o addirittura codificato nei materiali didattici e nelle tecnologie di supporto utilizzate. Questo non significa che una strada sia migliore di un'altra: stiamo semplicemente mettendo in evidenza il fatto che le caratteristiche più basilari di un determinato ambiente d'apprendimento ne determina fin dal principio tutto il suo andamento. In particolare, sembra che l'andamento orizzontale della paragogia si verifichi spesso all'interno della andragogia in particolar modo quando si invitano i partecipanti ad interagire fra loro in modo costruttivo e propositivo. In breve, siamo d'accordo con Blondy quando scrive che l'andragogia dovrebbe essere usata come punto di partenza e come primo approccio delle persone adulte in un ambiente educativo ideato on-line. Noi raccomandiamo la paragogia come ulteriore sviluppo in questa nuova dimensione dell'apprendimento condiviso.

5 Paragogia e la filosofia di Basho

IL PRIMO principio della paragogia sottolinea l'importanza nella comprensione del concetto condivisione in movimento. La filosofica attribuzione di questa nozione è attribuibile a Kitaro Nishida, e alla sua prima trascrizione in inglese curata da Masao Abe, in cui furono descritti i modi e gli eventi che permettono alle nuove nozioni emergono dai loro contesti. In altre parole l'idea di Basho, (condivisione del contenuto in movimento) ci aiuta a capire come i contesti educativi contengano e supportino al suo interno le nozioni (sub-action) e il modo in cui vengono reinterpretate nel contesto in cui si troviamo. Nonaka e Toyama prendono questa idea e la adottano alla creazione di nuovo sapere. In particolare, ci suggeriscono che la conoscenza si sviluppa dalla interazione delle persone in un contesto condiviso suddiviso nelle sue fasi. Questo sistema viene chiamato –SECI– *Socialisation, Externalisation, Combination, Internalisation*. (Socializzazione - Internalizzazione - Combinazione - Internazionalizzazione). In altre parole il primo concetto di paragogia può essere meglio compreso in questi termini:

- Perché – **Why I do it.**
- Come – **How we do it.**
- Cosa – **What we do it.**

Il primo concetto paragogico suggerisce di concentrarsi su come imparare (autodidatta o con affiancamento pedagogico), chiedendosi innanzitutto cosa gli studenti siano effettivamente in grado di fare. Si noti che in questa fase di ricerca, il contributo degli studenti riveste un ruolo attivo per ridisegnare il contesto di apprendimento e l'individuazione degli schemi da seguire. Invece di limitarsi a dire semplicemente “*qui manca la comprensione e propensione necessaria allo studio, quindi devo aiutarli in qualche modo*”, uno paralogogo guarderebbe anche le caratteristiche contestuali ambientali che ne limitano l'auto-apprendimento. Questi sono aspetti troppo spesso trascurati che bloccano la capacità degli studenti a modificare la propria metodologia di studio e limitano la possibilità di saper farsi aiutare.

6 Paragogia e Peer Production

IL LEGAME tra paragogia e produzione condivisa risalta ancor più il valore delle persone. Phillip Schmidt scrive:

«Upon closer inspection of commons-based peer production communities, we find learning at their core»

~

«Dopo un esame più attento alle comunità commons-based peer production, abbiamo bisogno di imparare dal loro cuore»

In "Education and Mind in the Knowledge Age" Carl Bereiter scrive:

Schools are places where knowledge creation can go on, but where it does not have to be market driven or competitive. [...] Knowledge creation in schools is the creation of knowledge by students for their own use. [...] To the extent that knowledge created in schools has value beyond the classroom where it was created, it enters into a barter economy.

Le scuole sono luoghi in cui la creazione di conoscenza si sviluppa, ma non deve essere orientata a un mercato competitivo. [...] la creazione di questo sapere è prerogativa dagli studenti a loro uso.[...]Nella misura in cui le conoscenze create nelle scuole ha valore al di là del classe in cui è stata creata, essa entra in un'economia di condivisione.

Contesto come centro decentrato: L'idea che la motivazione interna sia in conflitto con l'obiettivo di direzionalità di Tennant [12], appare inconsistente se si considera il reciproco effetto di miglioramento personale nei contesti ambientali descritti da Benkler e Nissenbaum [13].

Meta-learning come fonte di conoscenza: Continuando su questa idea l'acquisizione di competenze, occupabilità, e una buona reputazione, sembrano essere una semplice dinamica di auto-orientamento al fine migliorare la propria qualità di vita. In realtà, anche queste motivazioni hanno una loro ben precisa origine. In un'analisi di apprendimento corretta durante la produzione di contenuti, dovremmo sempre chiederci: cosa vogliamo apprendere? perché scegliamo questo tipo di metodologia d'apprendimento in particolare?

I collaboratori sono uguali e differentemente unici: Benkler descrive tre caratteristiche necessarie per la produzione condivisa:

Distribuito in Creative Commons CC0 1.0 universe.

6 PARAGOGIA E PEER PRODUCTION

1. Gli oggetti potenziali di produzione condivisa devono essere modulari.
2. i moduli devono essere di piccole dimensioni (notare che la distribuzione eterogenea, consentirà a persone con diversi livelli motivazionali di collaborare in modi diversi a piccoli o più significativi contributi).
3. il meccanismo di integrazione deve essere possibile ad un costo relativamente basso; (sia attraverso l'automazione tecnologica sia con l'attuazione di nuove norme sociali rafforzate). In paragogia, la scelta di lavorare in un piccolo gruppo chiuso (come descritto ad esempio in [14]) si evolve nella scelta di lavorare in un più ampio gruppo incorporato all'interno di differenti comunità (come descritto per esempio in [15]) incontrando la domanda: Con quante differenze sono disposto a confrontarti durante mio processo d'apprendimento?

L'apprendimento è distribuito e non lineare. La visione di un più avanzato contesto sociale proposto da Engestrom [3] come una movimento al di là delle tradizionali " comunità di interesse specifico " è del tutto compatibile con la virtù dei più famosi colleghi impegnati nella di produzione di nuove libertà (cfr. [13]), ciò, che permette alle persone di funzione in modo distributivo e non lineare in un "ecosistema" d'apprendimento e produzione di materiali nuovo. Star e Griesemer [16], su cui Wenger ha pesantemente contribuito durante lo sviluppando del concetto di "comunità di pratica" [17], descrivono la loro idea come "ecologica". Una chiave interpretativa tra Star/Wegner, da un lato e Engestrom, dall'altro, ha a che fare con la natura dei confini. Nella comunità di vista pratica, gli "oggetti di contorno" esistono per lo spostamento a nuove correnti di pensiero dei sui partecipanti diverse dalla idea originale d'insieme. Il punto di vista di Engestrom a riguardo, si richiama, attenzione dei confini indefiniti (attraverso un processo di co-formazione) o quando i limiti non sono ben definiti, (ad esempio nel caso in cui vi sia un annebbiamento delle regole tra consumatori e produttore). Un'idea strettamente connessa da Engestrom ruota attorno al concetto sociale di "oggetti condivisi", al posto di "connessioni astratte tra persone". (cfr. [19]). Combinando questo con l'idea di Basho, arriviamo alla immediata e intuitiva idea di sovrapposizione in contesti ed ambienti più ampi di "occasioni condivise". In un ambiente co-creato dai suoi partecipanti, è probabile che diventi un particolare e significativo luogo di nuove idee e valori.

Distribuito in Creative Commons CC0 1.0 universe.

Realizzare il sogno, poi risvegliarsi Definizioni contrastanti, rendono universalmente molto difficile la comprensione e la definizione di realizzazione. Tuttavia, come sottolinea Schmidt, quasi tutto è “misurabile”, ad esempio il sistema di valutazione di partecipazione nella produzione e scrittura nei progetti Open Software e open source [10]. Allo stesso modo, ci aspettiamo di saper misurare i simili contributi modulari applicati al ad commons-Peer su prodotti manifatturieri.[18]. Questo risulta essere una attività più impegnativa ma altrettanto necessaria al contributo alla l'integrazione e coordinamento.

7 Paragogia e Apprendimento Analitico

Veniamo ora alla applicazione principale della paragogia, cioè, produrre uno schema capace di dare forma a infinite ed efficaci possibilità di apprendimento analitico [d'ora in poi, **LA (Learning Analytics)**].

Contesto come centro decentrato. George Siemens definisce l'apprendimento analitico come l'uso di dati intelligenti, prodotti dagli studenti come informazioni e modelli di analisi per scoprire informazioni e connessioni sociali, utili alla previsione e al consiglio in materia di apprendimento. [20] Misurare il progresso di uno studente in un ambiente educativo dato, in caso centralizzato (Freshman di Senior) o decentrato (Jedi Padawan a), deve adeguatamente indicarne anche il contesto in ogni sua fase di sviluppo. Il progresso, può essere definito relativo rispetto alle attività dei suoi partecipanti e all' ambiente in cui si trova. In un caso semplice, LA costruirà e manterrà un sistema di raccolta dati e raggiungimento obiettivi monitorando i cambiamenti definiti da un istruttore o facilitatore. LA riveste una parte non banale di ogni contesto di apprendimento, suggerendo la trasparenza nelle sue diverse fasi, e il modo in cui saranno trattate tutte le informazioni utili da considerare. Proseguendo, lo studente e l'istruttore aumenteranno e concorderanno assieme l'efficienza delle metodologie da seguire. Un buon esempio, è il progetto formativo di Paul J. Williams per fornire agli studenti e alle scuole un modello funzionale, organizzativo, e decisionale che sia in grado di fare risparmiare tempo e risorse economiche imparando a distinguere tra “*priorità concorrenti e discordanti*”. Le istituzioni in questo modo hanno un inoltre sistema di riscontro per monitorare i loro investimenti in tecnologia, insegnanti, strutture, riscontrando efficacemente i miglioramenti di apprendimento effettivo di ogni studente. Mentre le applicazioni di LA basate su test standardizzati è oggi importante, si

7 PARAGOGIA E APPRENDIMENTO ANALITICO

spera che un maggiore studio in questo campo permetterà l'ideazione di test più sofisticati che consentiranno una valutazione più olistica dei miglioramenti di apprendimento . In particolare, si pone una domanda provocatoria circa il miglior sistema di misurazione dei successi scolastici. Bereiter, sottolinea lo sviluppo di un contesto che includa un aiuto funzionale agli studenti, diverso dalla semplice ricezione di nozioni, saggezza o incorporazione di esperienze esterne. Crede che pensando ad alta voce sia un modo di ricercare e capire come le persone realmente pensino ([11], p. 348). La Paragogia suggerisce una visione ancora più ampia di questa visione: invece di trovarsi in contesto didattico tradizionale, in una rete peer-based, la parola scorre in una rete di molto vasta, e il suo pensiero rappresenta un nuovo modello intrinsecamente sociale di studio.

Meta-learning come fonte di conoscenza: Un'altra definizione data è EDUCAUSE . Le prossime iniziative di apprendimento prevederanno l'uso di dati e modelli per predire il progresso degli studenti e le loro prestazioni, ottimizzando e la capacità di agire su tali informazioni. In breve, il meta-learning è il principio e l'applicazione più ovvia di LA: il modo più efficace di imparare sia non smettere di imparare mai.

I collaboratori sono uguali e differentemente unici: LA può essere non solo utilizzata per misurare i successi, i fallimenti, o le ore investite da un singolo studente, ma può risultare utile applicata in relazione ai dati tra collaboratori, tra i facilitatori o insegnanti (ad esempio per suggerimenti o critiche). LA potrebbe essere utilizzata per accoppiare gli studenti più deboli con quelli più avanzati, o per aiutare gli studenti con interessi simili a unirsi e mettersi in evidenza all'interno del gruppo. I valori di soglia possono essere impostati per indicare quando uno studente sia pronto o richiesto di passare a un ruolo di mentore o di monitoraggio. Le misurazioni possono essere effettuate per rilevare il livello di collaborazione attiva all'interno del gruppo di lavoro , o singolarmente per il miglioramene l'ambiente di apprendimento. I dati su come i vari profili di studenti assimilino meglio, possono essere combinati dal facilitatore al fine di migliorare l'intero contesto educativo . Altre valide raccomandazioni sono parte in oggetto di un corpus significativo di ricerche in attualmente in corso.

Distribuito in Creative Commons CC0 1.0 universe.

7 PARAGOGIA E APPRENDIMENTO ANALITICO

L'apprendimento è distribuito e non lineare: LA, in particolare l'attenzione dei metadati, è in grado di misurare la quantità e la qualità di uno studente nel rimanere focalizzato su un determinato argomento, fornendo feedback utili nel tempo a breve e lungo periodo . Quando si accede al una classe wiki , lavoreranno alla pagina per un'ora intera , o si distrarranno ? Quando uno studente incomincia e poi abbandona bisognerebbe sempre tenerne traccia al fine di comprenderne le motivazioni che potranno risultare utili in seguito. . Gli studenti possono mostrare il loro apprendimento in settori anche di non propria competenza . Ad esempio, un laureato in letteratura straniera che decidesse di cambiare carriera presentando il proprio profilo a una industria meccanica, potrebbe indipendentemente avere completato la maggior parte del percorso formativo in giornalismo e avere delle forti motivazioni o un profilo interessante per lavorare come addetto alle relazioni estere per questa azienda. Per arrivare al punto in cui il sistema può dare un feedback di questa natura, gli obiettivi devono essere specificata e concordati in anticipatamente. Gli obbiettivi a lungo termine sono generalmente abbastanza facili da individuare , ad esempio "Voglio imparare il giapponese". E 'più difficile spezzare un compito in passi, e il primo è spesso il più difficile. Corneli ha suggerito di cercare il più semplice (ciò si può effettivamente fare) che vi porta verso il vostro biettivo. Gli studenti possono quindi condividere quel passo per quanto piccolo, e una volta raggiunto, può scegliere un altro lungo la strada. Questi modelli possono essere studiati per trovare LA che mostri la percentuale di miglioramento ad esempio nella fluidità espositiva in giapponese.

Realizzare il proprio perché e poi viverlo! Riteniamo che questa sia è la chiave che unisce l'apprendimento analitico e la paragogia: uno studente dovrebbe esplicitamente dichiarare le sue motivazioni/obiettivi e poi tenere traccia dei propri progressi verso il raggiungimento alla meta. LA aiuterà gli studenti a valutare un modo chiaro i prossimi passi necessari a realizzare le loro aspettative, mostrare i risultati ottenuti al resto del mondo. In caso di problemi, LA dovrebbe aiutare a identificare i cambiamenti che possono aiutare. E pensare che qui evidenziare una possibile applicazione su vasta scala, possiamo immaginare la creazione di standard di accreditamento paragogici per studenti, simili ad esempio a quelli utilizzati oggi per le imprese di Better Business Bureau. Questo potrebbe venire da un sistema capace di tenere traccia dei tipi di corsi che più interessano e inoltre, offrire le risorse e gli strumenti utili al

Distribuito in Creative Commons CC0 1.0 universe.

raggiungimento di una specifica conoscenza e livello di impegno prima dell'inizio del corso. Il grado in cui le persone rispettano i propri impegni nel tempo, avrebbe inoltre una influenza sulla la loro valutazione e credibilità.

8 Conclusioni

Abbiamo esplorato le connessioni tra la paragogia e la produzione condivisa e tra paragogia e apprendimento analitico mostrando come questo nuovo approccio possa aprire nuove frontiere alla produzione di nuovi contenuti educativi

8.1 Prossimi passi degli autori

Entrambi svilupperemo attraverso i nuovi corsi della Università Peer 2 Peer le attività paragogiche e la sua evoluzione. Un'altra possibilità che stiamo esplorando, è la creazione di un nostro profilo studente come suggerito da Siemens e costruendo un framework di criteri e implementazioni al modello. Faremo inoltre il possibile per incoraggiare chiunque voglia approfondire le tematiche in questione

8.2 Implementando la Paragogia

Noi incoraggiamo tutte le comunità di ricerca a testare la nostra idea in modi pratici e differenti. Alcuni punti possono essere:

1. Stabilire un gruppo di valutazione per le aspettative, gli obiettivi che valuteranno le conclusioni.
2. Aiutare un gruppo di studenti a definire i propri obiettivi e affiancarli.
3. Formalizzare un processo di assistenza condivisa (Es. rispondere alle domande, offrire un riscontro e diffondere il lavoro svolto).
4. Costruire in modo pratico una metodologia interna per applicare gli insegnamenti nei propri contesti educativi.

Distribuito in Creative Commons CC0 1.0 universe.

Ringraziamenti:

1. Jonathan Grudin: Perché le applicazioni non CSCW: problemi nella progettazione e valutazione delle interfacce organizzative. Atti del convegno 1988 ACM su Computer-Supported Cooperative Work, pp 85 93 (1988).
2. Mary-Jane Eisen: Peer-Based Learning: un nuovo-vecchio alternativa allo sviluppo professionale. Adult Learning, 12 (1) pp 9 10 (2001).
3. Y. Engestrom: Dalla comunità di pratica di mycorrhizae, in H. Hughes, N.Jewson, L. Unwin (a cura di), Le comunità di pratica: prospettive critiche. London: Routledge (2007).

Distribuito in Creative Commons CC0 1.0 universe.

<p>Charles Jeffrey Danoff My talk My preferences My watchlist My contributions Log out Page Discussion edit History delete Move Protect Watch Zotero group</p> <div><input type="text"/> <input type="button" value="Go"/> <input type="button" value="SEARCH"/></div>	
<p><u>Main page</u> <u>Recent changes</u> <u>Issue Tracker</u></p> <p><u>What links here</u> <u>Related changes</u> <u>Upload file</u> <u>Special pages</u> <u>Printable version</u> <u>Permanent link</u></p>	<h2>ENG 099 CONVERSATIONAL AMERICAN ENGLISH TEXTBOOK 1.1ST ED.</h2> <p>ENG 099 Conversational American English Textbook 1.1st e-book Edition</p> <p>by Charles Jeffrey Danoff</p> <p>Published by Pub Dom Ed Press/Paragogy.net</p> <p><i>Pub Dom Ed Press is an imprint of Mr. Danoff's Teaching Laboratory (http://mr.danoff.org) .</i></p> <p>The first e-book edition (http://is.gd/ENG099textbook1) of ENG 099 Conversational American English Textbook was published on September 8, 2011 by Mr. Danoff's Teaching Laboratory/Paragogy.net^[1].</p> <p>Attention: Pub Dom Ed Press Mr. Danoff's Teaching Laboratory Post Office Box 612 Winnetka, Illinois 60093-0612 United States of America</p> <p>+1.315.750.9903 contact@mr.danoff.org http://mr.danoff.org</p> <p>Copyright (C) 2011, 2012 Charles Jeffrey Danoff. Ownership rights abandoned by author. Everything in this textbook is in the public domain. Re-published parts not authored by Charles are taken from resources that are in the public domain in the USA. N.B. this work does also include links to non-public domain works.</p> <div><h2>GREETINGS</h2><h3><i>Lesson Plan</i></h3><p>Opening Introduce myself to new students, talk about</p></div> <div><h3>Contents</h3><p>1 Greetings 1.1 Lesson Plan 1.2 AAR</p></div>

<p>their summer and answer student questions.</p> <p>Topic This week's lesson is about American English greetings, as well as setting up the rest of the 10 week course.</p> <p>Goals Students write on board, "My goal today is _____."</p>	<p>2 Formal Telephone English 2.1 Lesson Plan 2.2 AAR</p> <p>3 Informal Telephone English 3.1 Lesson Plan 3.2 AAR</p> <p>4 Restaurant Menus 4.1 Lesson Plan 4.2 AAR</p> <p>5 Government Forms 5.1 Lesson Plan 5.2 AAR</p> <p>6 Reading American Fiction 6.1 Lesson Plan 6.2 AAR</p> <p>7 Writing American English Primer 7.1 Lesson Plan 7.2 AAR</p> <p>8 The Job Interview 1 8.1 Lesson Plan 8.2 AAR</p> <p>9 The Job Interview 2 9.1 Lesson Plan 9.2 AAR</p> <p>10 Review 10.1 Lesson Plan 10.2 AAR</p> <p>11 Notes</p>
<hr/> <p>Greetings Elicit what students know about American greetings, show video^[2] on friendly greetings and then go over slowly with the slideshow^[3].</p> <p>Class Overview Go through how the class will work and answer any questions.</p> <p>Language Talk^[4]</p> <p>+Teacher+.—I will pronounce these three sounds very slowly and distinctly, thus: b-u-d. Notice, it is the power, or sound, of the letter, and not its name, that I give. What did you hear?</p> <p>+Pupil+.—I heard three sounds.</p> <p>+T.—+Give them. I will write on the board, so that you can see them, three letters—b-u-d. Are these letters, taken separately, signs to you of anything?</p> <p>+P.—+Yes, they are signs to me of the three sounds that I have just heard.</p> <p>+T.—+What then do these letters, taken separately, picture to your eye?</p> <p>+P.—+They picture the sounds that came to my ear.</p> <p>+T+.—Letters then are the signs of what?</p> <p>+P.—Letters are the signs of sounds+.</p> <p>+T+.—I will pronounce the same three sounds more rapidly, uniting them more closely—bud. These sounds, so united, form a spoken word. Of what do you think when you hear the word bud?</p> <p>+P+.—I think of a little round thing that grows to be a leafy branch or a flower.</p> <p>+T+.—Did you see the thing when you were thinking of it?</p> <p>+P+.—No.</p> <p>+T+.—Then you must have had a picture of it in your mind. We call this +mental picture+ an +idea+. What called up this idea?</p> <p>+P+.—It was called up by the word bud, which I heard.</p> <p>+T+.—A spoken word then is the sign of what?</p> <p>+P.—A spoken word is the sign of an idea+.</p>	

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"

ENG 099 Conversational American English Textbook 1.1st Ed. - Paragogy.net http://paragogy.net/index.php?title=ENG_099_Conversational_American...

+T+.—I will call up the same idea in another way. I will write three letters and unite them thus: bud. What do you see?
+P+.—I see the word bud.
+T+.—If we call the other word bud a spoken word, what shall we call this?
+P+.—This is a written word.
+T+.—If they stand for the same idea, how do they differ?
+P+.—I see this, and I heard that.
+T+.—You will observe that we have called attention to four different things; viz., the +real bud+; your mental picture of the bud, which we have called an +idea+; and the +two words+, which we have called signs of this idea, the one addressed to the ear, and the other to the eye.

Tom Sawyer^[5] Tom Sawyer is a famous American novel by Mark Twain. Each week we will read some of the story and talk about the work.

"TOM!"

No answer.

"TOM!"

No answer.

"What's gone with that boy, I wonder? You TOM!"

No answer.

The old lady pulled her spectacles down and looked over them about the room; then she put them up and looked out under them. She seldom or never looked THROUGH them for so small a thing as a boy; they were her state pair, the pride of her heart, and were built for "style," not service—she could have seen through a pair of stove-lids just as well. She looked perplexed for a moment, and then said, not fiercely, but still loud enough for the furniture to hear:

"Well, I lay if I get hold of you I'll—"

Please write 2 sentences with your opinion on what you just read.

Next Time Find out if there is anything specific the students want to learn about next week.

Extra Time Student questions, game or student directed activities.

AAR

AAR stands for After Action Review, idea taken from [<http://www.army.mil/features/FM7/FM%207-0.pdf> Training the Force (FM-07)] by the US Army (2002-10-22).

Review what was supposed to happen.

-

Establish what happened.

-

Determine what was right or wrong with what happened.

-

Determine how the task should be done differently the next time.

-

FORMAL TELEPHONE ENGLISH

Lesson Plan

Opening Chat about our past week and answer any questions students have.

Topic This week's lesson is about formal telephone English.

Goals Students write on board, "My goal today is _____."

Formal Telephone English

- Watch Learn English 4-2 : Answering the Phone YouTube video^[6], stopping frequently and discussing.
- Discuss About.com's Telephone Conversations^[7] and Englishclub.com's^[8] Telephone tips page.

Language Talk^[4]

+Teacher+.—What did you learn in the previous Lesson?

+Pupil+.—I learned that a spoken word is composed of certain sounds, and that

letters are signs of sounds, and that spoken and written words are the signs of ideas.

This question should be passed from one pupil to another till all of these answers are elicited.

All the written words in all the English books ever made, are formed of twenty-six letters, representing about forty sounds. These letters and these sounds make up what is called artificial language.

Of these twenty-six letters, +a, e, i, o, u+, and sometimes +w+ and +y+, are called +vowels+, and the remainder are called +consonants+.

In order that you may understand what kind of sounds the vowels stand for, and what kinds the consonants represent, I will tell you something about the human voice.

The air breathed out from your lungs beats against two flat muscles, stretched like strings across the top of the windpipe, and causes them to vibrate. This vibrating makes sound. Take a thread, put one end between your teeth, hold the other in your fingers, draw it tight and strike it, and you will understand how voice is made.

If the voice thus produced comes out through the mouth held well open, a class of sounds is formed which we call vowel sounds.

But, if the voice is held back by your palate, tongue, teeth, or lips, one kind of consonant sounds is made. If the breath is driven out without voice, and is held back by these same parts of the mouth, the other kind of consonant sounds is formed. Ex. of both: b, d, g; p, t, k.

The teacher and pupils should practice on these sounds till the three kinds can easily be distinguished.

You are now prepared to understand what I mean when I say that the +vowels+ are the +letters+ which stand for the +open sounds of the voice+, and that the +consonants+ are the +letters+ which stand for the sounds made by the +obstructed voice+ and the +obstructed breath+.

The teacher can here profitably spend a few minutes in showing how ideas may be communicated by Natural Language, the language of sighs, groans, gestures of the hands, attitudes of the body, expressions of the face, tones of the voice, etc. He can show that, in conversation, we sometimes couple this Natural Language of tone and gesture with our language of words, in order to make a stronger impression. Let the pupil be told that, if the passage contain feeling, he should do the same in Reading and Declaiming.

Let the following definitions be learned, and given at the next recitation.

+DEFINITION.—Artificial Language, or Language Proper, consists of the spoken and written words used to communicate ideas and thoughts+.

+DEFINITION.—English Grammar is the science which teaches the forms, uses, and relations of the words of the English Language+.

Tom Sawyer^[5]

She did not finish, for by this time she was bending down and punching under the bed with the broom, and so she needed breath to punctuate the punches with. She resurrected nothing but the cat.

"I never did see the beat of that boy!"

She went to the open door and stood in it and looked out among the tomato vines and "jimpson" weeds that constituted the garden. No Tom. So she lifted up her voice at an angle calculated for distance and shouted:

"Y-o-u-u TOM!"

There was a slight noise behind her and she turned just in time to seize a small boy by the slack of his roundabout and arrest his flight.

"There! I might 'a' thought of that closet. What you been doing in there?"

"Nothing."

"Nothing! Look at your hands. And look at your mouth. What IS that truck?"

"I don't know, aunt."

"Well, I know. It's jam—that's what it is. Forty times I've said if you didn't let that jam alone I'd skin you. Hand me that switch."

Please write 2 sentences with your opinion on what you just read.

Next Time Find out if there is anything specific the students want to learn about next week.

Extra Time Student questions, game or student directed activities.

AAR

Review what was supposed to happen.

■

Establish what happened.

▪

Determine what was right or wrong with what happened.

▪

Determine how the task should be done differently the next time.

INFORMAL TELEPHONE ENGLISH

Lesson Plan

Opening Chat about our past week and answer any questions students have.

Topic This week's lesson is about informal telephone English.

Goals Students write on board, "My goal today is _____."

Informal Telephone English For a check-in call to a friend nearby, i.e. "Wacha doin'?" Go through an example call made between two American friends and have the students act it out.

Language Talk^[4]

Let the pupils be required to tell what they learned in the previous lessons.

+Teacher+.—When I pronounce the two words star and bud thus: star bud, how many ideas, or mental pictures, do I call up to you?

+Pupil+.—Two.

+T+.—Do you see any connection between these ideas?

+P+.—No.

+T+.—When I utter the two words bud and swelling, thus: bud swelling, do you see any connection in the ideas they stand for?

+P+.—Yes, I imagine that I see a bud expanding, or growing larger.

+T+.—I will connect two words more closely, so as to express a thought: Buds swell. A thought has been formed in my mind when I say, Buds swell; and these two words, in which something is said of something else, express that thought, and make what we call a sentence. In the former expression, bud swelling it is assumed, or taken for granted, that buds perform the act; in the latter, the swelling is asserted as a fact.

Leaves falling. Do these two words express two ideas merely associated, or do they express a thought?

+P+.—They express ideas merely associated.

+T+.—Leaves fall.

Same question.

+P+.—A thought.

+T+.—Why?

+P+.—Because, in these words, there is something said or asserted of leaves.

+T+.—When I say, Falling leaves rustle, does falling tell what is thought of leaves?

+P+.—No.

+T+.—What does falling do?

+P+.—It tells the kind of leaves you are thinking and speaking of.

+T+.—What word does tell what is thought of leaves?

+P+.—Rustle.

+T+.—You see then that in the thought there are two parts; something of which we think, and that which we think about it.

Let the pupils give other examples.

Tom Sawyer^[5]

The switch hovered in the air—the peril was desperate—

"My! Look behind you, aunt!"

The old lady whirled round, and snatched her skirts out of danger. The lad fled on the instant, scrambled up the high board-fence, and disappeared over it.

His aunt Polly stood surprised a moment, and then broke into a gentle laugh.

Please write 3 sentences with your opinion on what you just read.

Next Time Find out if there is anything specific the students want to learn about next week.

Extra Time Student questions, game or student directed activities.

AAR

Review what was supposed to happen.

-

Establish what happened.

-

Determine what was right or wrong with what happened.

-

Determine how the task should be done differently the next time.

RESTAURANT MENUS

Lesson Plan

Opening Chat about our past week and answer any questions students have.

Topic This week's lesson is about restaurant menus.

Goals Students write on board, "My goal today is _____."

Restaurant Menus Elicit how to order at restaurants with menus from local places and/or fast food joints.

Language Talk^[4]

Commit to memory all definitions.

+DEFINITION.—A Sentence is the expression of a thought in words+.

Which of the following expressions contain words that have no connection, which contain words merely associated, and which are sentences?

1. Flowers bloom. 2. Ice melts. 3. Bloom ice. 4. Grass grows. 5. Brooks babble. 6. Babbling brooks. 7. Grass soar. 8. Doors open. 9. Open doors. 10. Cows graze. 11. Curling smoke. 12. Sugar graze. 13. Dew sparkles. 14. Hissing serpents. 15. Smoke curls. 16. Serpents hiss. 17. Smoke curling. 18. Serpents sparkles. 19. Melting babble. 20. Eagles soar. 21. Birds chirping. 22. Birds are chirping. 23. Birds chirp. 24. Gentle cows. 25. Eagles are soaring. 26. Bees ice. 27. Working bees. 28. Bees work. 29. Crawling serpents. 30. Landscape piano. 31. Serpents crawl. 32. Eagles clock. 33. Serpents crawling.

Tom Sawyer^[5]

"Hang the boy, can't I never learn anything? Ain't he played me tricks enough like that for me to be looking out for him by this time? But old fools is the biggest fools there is. Can't learn an old dog new tricks, as the saying is. But my goodness, he never plays them alike, two days, and how is a body to know what's coming? He 'pears to know just how long he can torment me before I get my dander up, and he knows if he can make out to put me off for a minute or make me laugh, it's all down again and I can't hit him a lick. I ain't doing my duty by that boy, and that's the Lord's truth, goodness knows. Spare the rod and spile the child, as the Good Book says. I'm a laying up sin and suffering for us both, I know. He's full of the Old Scratch, but laws-a-me! he's my own dead sister's boy, poor thing, and I ain't got the heart to lash him, somehow. Every time I let him off, my conscience does hurt me so, and every time I hit him my old heart most breaks. Well-a-well, man that is born of woman is of few days and full of trouble, as the Scripture says, and I reckon it's so. He'll play hookey this evening, * and [* Southwestern for "afternoon"] I'll just be obleeged to make him work, tomorrow, to punish him. It's mighty hard to make him work Saturdays, when all the boys is having holiday, but he hates work more than he hates anything else, and I've GOT to do some of my duty by him, or I'll be the ruination of the child."

Please write 3 sentences with your opinion on what you just read.

Next Time Find out if there is anything specific the students want to learn about next week.

Extra Time Student questions, game or student directed activities.

AAR

Review what was supposed to happen.

▪

Establish what happened.

▪

Determine what was right or wrong with what happened.

▪

Determine how the task should be done differently the next time.

GOVERNMENT FORMS

Lesson Plan

Opening Chat about our past week and answer any questions students have.

Topic This week's lesson is about forms for the US Government.

Goals Students write on board, "My goal today is _____."

Government Forms Start with IRS Form 1040^[9] and go through it together as a class.

Language Talk^[4]

Illustrate, by the use of a, b, and p, the difference between the sounds of letters and their names. Letters are the signs of what? What is an idea? A spoken word is the sign of what? A written word is the sign of what? How do they differ? To what four different things did we call attention in Lesson 1?

How are vowel sounds made? How are the two kinds of consonant sounds made? What are vowels? Name them. What are consonants? What is artificial language, or language proper? What do you understand by natural language?

What is English grammar?

What three kinds of expressions are spoken of in Lessons 3 and 4? Give examples of each. What is a sentence?

Tom Sawyer^[5]

Tom did play hookey, and he had a very good time. He got back home barely in season to help Jim, the small colored boy, saw next-day's wood and split the kindlings before supper—at least he was there in time to tell his adventures to Jim while Jim did three-fourths of the work. Tom's younger brother (or rather half-brother) Sid was already through with his part of the work (picking up chips), for he was a quiet boy, and had no adventurous, trouble-some ways.

While Tom was eating his supper, and stealing sugar as opportunity offered, Aunt Polly asked him questions that were full of guile, and very deep—for she wanted to trap him into damaging revealments. Like many other simple-hearted souls, it was her pet vanity to believe she was endowed with a talent for dark and mysterious diplomacy, and she loved to contemplate her most transparent devices as marvels of low cunning. Said she:

Please write 4 sentences with your opinion on what you just read.

Next Time Find out if there is anything specific the students want to learn about next week.

Extra Time Student questions, game or student directed activities.

AAR

Review what was supposed to happen.

▪

Establish what happened.

▪

Determine what was right or wrong with what happened.

▪

Determine how the task should be done differently the next time.

READING AMERICAN FICTION

Lesson Plan

Opening Chat about past week and answer any questions students have.

Topic This week's lesson is about how to read American fiction.

Goals Students write on board, "My goal today is _____."

Reading American Fiction Vladimir Nabokov is a famous Russian novelist who taught at American Universities in the 20th century. We will read some of his lecture "Good Readers and Good Writers" ^[10] to think about how we can read American fiction.

Language Talk^[4]

On the following sentences, let the pupils be exercised according to the model.

+Model+.—Intemperance degrades. Why is this a sentence? Ans.—Because it expresses a thought. Of what is something thought? Ans.—Intemperance. Which word tells what is thought? Ans.—Degrades.

1. Magnets attract. 2. Horses neigh. 3. Frogs leap. 4. Cold contracts. 5. Sunbeams dance. 6. Heat expands. 7. Sunlight gleams. 8. Banners wave. 9. Grass withers. 10. Sailors climb. 11. Rabbits burrow. 12. Spring advances.

You see that in these sentences there are two parts. The parts are the +Subject+ and the +Predicate+.

+DEFINITION.—The Subject of a sentence names that of which something is thought+.

+DEFINITION.—The Predicate of a sentence tells what is thought+.

+DEFINITION.—The Analysis of a sentence is the separation of it into its parts+.

Analyze, according to the model, the following sentences.

+Model+.—Stars twinkle. This is a sentence, because it expresses a thought. Stars is the subject, because it names that of which something is thought; twinkle is the predicate, because it tells what is thought.

+To the Teacher+.—After the pupils become familiar with the definitions, the "Models" may be varied, and some of the reasons maybe made specific; as, "Plants names the things we tell about; droop tells what plants do," etc.

Guard against needless repetition.

1. Plants droop. 2. Books help. 3. Clouds float. 4. Exercise strengthens. 5. Rain falls. 6. Time flies. 7. Rowdies fight. 8. Bread nourishes. 9. Boats capsize. 10. Water flows. 11. Students learn. 12. Horses gallop.

Tom Sawyer^[5]

"Tom, it was middling warm in school, warn't it?"

"Yes'm."

"Powerful warm, warn't it?"

"Yes'm."

"Didn't you want to go in a-swimming, Tom?"

A bit of a scare shot through Tom—a touch of uncomfortable suspicion. He searched Aunt Polly's face, but it told him nothing. So he said:

"No'm—well, not very much."

The old lady reached out her hand and felt Tom's shirt, and said:

"But you ain't too warm now, though." And it flattered her to reflect that she had discovered that the shirt was dry without anybody knowing that that was what she had in her mind. But in spite of her, Tom knew where the wind lay, now. So he forestalled what might be the next move:

"Some of us pumped on our heads—mine's damp yet. See?"

Aunt Polly was vexed to think she had overlooked that bit of circumstantial evidence, and missed a trick. Then she had a new inspiration:

"Tom, you didn't have to undo your shirt collar where I sewed it, to pump on your head, did you? Unbutton your jacket!"

Please write 4 sentences with your opinion on what you just read.

Next Time Find out if there is anything specific the students want to learn about next week.

Extra Time Student questions, game or student directed activities.

AAR

Review what was supposed to happen.

-

Establish what happened.

-

Determine what was right or wrong with what happened.

-

Determine how the task should be done differently the next time.

WRITING AMERICAN ENGLISH PRIMER

Lesson Plan

Opening

Topic This week we are going to introduce how to write American English.

Goals Students write on board, "My goal today is _____."

Writing American English Primer The book **The Elements of Style** ^[11] by William Strunk is one of the most famous books on writing American English. We will read a short section and talk about it.

Make the paragraph the unit of composition: one paragraph to each topic.

If the subject on which you are writing is of slight extent, or if you intend to treat it very briefly, there may be no need of subdividing it into topics. Thus a brief description, a brief summary of a literary work, a brief account of a single incident, a narrative merely outlining an action, the setting forth of a single idea, any one of

these is best written in a single paragraph. After the paragraph has been written, examine it to see whether subdivision will not improve it.

Ordinarily, however, a subject requires subdivision into topics, each of which should be made the subject of a paragraph. The object of treating each topic in a paragraph by itself is, of course, to aid the reader. The beginning of each paragraph is a signal to him that a new step in the development of the subject has been reached.

The extent of subdivision will vary with the length of the composition. For example, a short notice of a book or poem might consist of a single paragraph. One slightly longer might consist of two paragraphs:

A. Account of the work. B. Critical discussion.

A report on a poem, written for a class in literature, might consist of seven paragraphs:

A. Facts of composition and publication. B. Kind of poem; metrical form. C. Subject. D. Treatment of subject. E. For what chiefly remarkable. F. Wherein characteristic of the writer. G. Relationship to other works.

The contents of paragraphs C and D would vary with the poem. Usually, paragraph C would indicate the actual or imagined circumstances of the poem (the situation), if these call for explanation, and would then state the subject and outline its development. If the poem is a narrative in the third person throughout, paragraph C need contain no more than a concise summary of the action. Paragraph D would indicate the leading ideas and show how they are made prominent, or would indicate what points in the narrative are chiefly emphasized.

A novel might be discussed under the heads:

A. Setting. B. Plot. C. Characters. D. Purpose.

An historical event might be discussed under the heads:

A. What led up to the event. B. Account of the event. C. What the event led up to.

In treating either of these last two subjects, the writer would probably find it necessary to subdivide one or more of the topics here given.

As a rule, single sentences should not be written or printed as paragraphs. An exception may be made of sentences of transition, indicating the relation between the parts of an exposition or argument. Frequent exceptions are also necessary in textbooks, guidebooks, and other works in which many topics are treated briefly.

In dialogue, each speech, even if only a single word, is a paragraph by itself; that is, a new paragraph begins with each change of speaker. The application of this rule, when dialogue and narrative are combined, is best learned from examples in well-printed works of fiction.

Tom Sawyer^[5]

The trouble vanished out of Tom's face. He opened his jacket. His shirt collar was securely sewed.

"Bother! Well, go 'long with you. I'd made sure you'd played hookey and been a-swimming. But I forgive ye, Tom. I reckon you're a kind of a singed cat, as the saying is—better'n you look. THIS time."

She was half sorry her sagacity had miscarried, and half glad that Tom had stumbled into obedient conduct for once.

But Sidney said:

"Well, now, if I didn't think you sewed his collar with white thread, but it's black."

"Why, I did sew it with white! Tom!"

But Tom did not wait for the rest. As he went out at the door he said:

"Siddy, I'll lick you for that."

In a safe place Tom examined two large needles which were thrust into the lapels of his jacket, and had thread bound about them—one needle carried white thread and the other black. He said:

"She'd never noticed if it hadn't been for Sid. Confound it! sometimes she sews it with white, and sometimes she sews it with black. I wish to gee-miny she'd stick to one or t'other—I can't keep the run of 'em. But I bet you I'll lam Sid for that. I'll learn him!"

He was not the Model Boy of the village. He knew the model boy very well though—and loathed him.

Please write 5 sentences with your opinion on what you just read.

Next Time Find out if there is anything specific the students want to learn about next week.

Extra Time Student questions, game or student directed activities.

AAR

Review what was supposed to happen.

■

Establish what happened.

-

Determine what was right or wrong with what happened.

-

Determine how the task should be done differently the next time.

THE JOB INTERVIEW 1

Lesson Plan

Opening Chat about the past week and answer any student questions.

Topic Go over how to do a job interview with an American company.

Goals Students write on board, "My goal today is _____."

Job Interview Start by going over some basic interview questions:

- Tell us about yourself.
- Why do you want this job?
- Why should we hire you?
- What's a difficult situation you've overcome?
- What are your weaknesses?

Go over the interview video^[12] and answer student questions.

Language Talk^[4]

ANALYSIS AND THE DIAGRAM.

+Hints for Oral Instruction+.—I will draw on the board a heavy, or shaded, line, and divide it into two parts, thus:

We will consider the first part as the sign of the subject of a sentence, and the second part as the sign of the predicate of a sentence.

Now, if I write a word over the first line, thus—(doing it)—you will understand that that word is the subject of a sentence. If I write a word over the second line, thus—you will understand that that word is the predicate of a sentence. Planets | revolve

The class can see by this picture that Planets revolve is a sentence, that planets is the subject, and that revolve is the predicate.

These signs, or illustrations, made up of straight lines, we call

+Diagrams+.

+DEFINITION.—A Diagram is a picture of the offices and relations of the different parts of a sentence+.

Analyze and diagram the following sentences.

1. Waves dash. 2. Kings reign. 3. Fruit ripens. 4. Stars shine. 5. Steel tarnishes. 6. Insects buzz. 7. Paul preached. 8. Poets sing. 9. Nero fiddled. 10. Larks sing. 11. Water ripples. 12. Lambs frisk. 13. Lions roar. 14. Tigers growl. 15. Breezes sigh. 16. Carthage fell. 17. Morning dawns. 18. Showers descended. 19. Diamonds sparkle. 20. Alexander conquered. 21. Jupiter thunders. 22. Columbus sailed, 23. Grammarians differ. 24. Cornwallis surrendered.

Tom Sawyer^[5]

Within two minutes, or even less, he had forgotten all his troubles. Not because his troubles were one whit less heavy and bitter to him than a man's are to a man, but because a new and powerful interest bore them down and drove them out of his mind for the time—just as men's misfortunes are forgotten in the excitement of new enterprises. This new interest was a valued novelty in whistling, which he had just acquired from a negro, and he was suffering to practise it un-disturbed. It consisted in a peculiar bird-like turn, a sort of liquid warble, produced by touching the tongue to the roof of the mouth at short intervals in the midst of the music—the reader probably remembers how to do it, if he has ever been a boy. Diligence and attention soon gave him the knack of it, and he strode down the street with his mouth full of harmony and his soul full of gratitude. He felt much as an astronomer feels who has discovered a new planet—no doubt, as far as strong, deep, unalloyed pleasure is concerned, the advantage was with the boy, not the astronomer.

The summer evenings were long. It was not dark, yet. Presently Tom checked his whistle. A stranger was before him—a boy a shade larger than himself. A new-comer of any age or either sex was an im-pressive curiosity in the poor little shabby village of St. Petersburg. This boy was well dressed, too—well dressed on a week-day. This was simply as- tounding. His cap was a dainty thing, his close-buttoned blue cloth roundabout was new and natty, and so were his pantaloons. He had shoes on—and it was only Friday. He even wore a necktie, a bright bit of ribbon. He had a citified air about him that ate into Tom's vitals. The more Tom stared at the splendid marvel, the higher he turned up his nose at his finery and the shabbier and shabbier his own outfit seemed to him to grow.

Neither boy spoke. If one moved, the other moved—but only sidewise, in a circle; they kept face to face and eye to eye all the time. Finally Tom said:

"I can lick you!"

"I'd like to see you try it."

Please write 5 sentences with your opinion on what you just read.

Next Time Find out if there is anything specific the students want to learn about next week.

Extra Time Student questions, game or student directed activities.

AAR

Review what was supposed to happen.

-

Establish what happened.

-

Determine what was right or wrong with what happened.

-

Determine how the task should be done differently the next time.

THE JOB INTERVIEW 2

Lesson Plan

Opening Chat about our past weeks and answer any student questions.

Topic We will expand upon the job interview and practice.

Goals Students write on board, "My goal today is _____."

Job Interview Review what we did last week and have students do mock job interviews of each other.

Language Talk^[4]

SENTENCE-BUILDING.

You have now learned to analyze sentences, that is, to separate them into their parts. You must next learn to put these parts together, that is, to build sentences.

We will find one part, and you must find the other and do the building.

+To the Teacher+.—Let some of the pupils write their sentences on the board, while others are reading theirs. Then let the work on the board be corrected.

Correct any expression that does not make good sense, or that asserts something not strictly true; for the pupil should early be taught to think accurately, as well as to write and speak grammatically.

Correct all mistakes in spelling, and in the use of capital letters and the period.

Call attention to the agreement in form of the predicate with the subject. See Notes, p. 163.

Insist on neatness. Collect the papers before the recitation closes.

+CAPITAL LETTER-RULE.—The first word of every sentence must begin with a capital letter+.

+PERIOD—RULE.—A period must be placed after every sentence that simply affirms, denies, or expresses a command+.

Construct sentences by supplying a subject to each of the following predicates.

Ask yourself the question, What swim, sink, hunt, etc.?

1. — swim. 2. — sinks. 3. — hunt. 4. — skate. 5. — jingle. 6. — decay. 7. — climb. 8. — creep. 9. — run. 10. — walk. 11. — snort. 12. — kick. 13. — flashes. 14. — flutters. 15. — paddle. 16. — toil. 17. — terrifies. 18. — rages. 19. — expand. 20. — jump. 21. — hop. 22. — bellow. 23. — burns. 24. — evaporates.

This exercise may profitably be extended by requiring the pupils to supply several subjects to each predicate.

Tom Sawyer^[5]

"Well, I can do it."

"No you can't, either."

"Yes I can."

"No you can't."

"I can."

"You can't."

"Can!"

"Can't!"

An uncomfortable pause. Then Tom said:

"What's your name?"

"Tisn't any of your business, maybe."

"Well I 'low I'll MAKE it my business."

"Well why don't you?"

"If you say much, I will."

"Much—much—MUCH. There now."

"Oh, you think you're mighty smart, DON'T you? I could lick you with one hand tied behind me, if I wanted to."

"Well why don't you DO it? You SAY you can do it."

"Well I WILL, if you fool with me."

"Oh yes—I've seen whole families in the same fix."

"Smarty! You think you're SOME, now, DON'T you? Oh, what a hat!"

"You can lump that hat if you don't like it. I dare you to knock it off—and anybody that'll take a dare will suck eggs."

"You're a liar!"

"You're another."

"You're a fighting liar and dasn't take it up."

"Aw—take a walk!"

"Say—if you give me much more of your sass I'll take and bounce a rock off'n your head."

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"

ENG 099 Conversational American English Textbook 1.1st Ed. - Paragogy.net http://paragogy.net/index.php?title=ENG_099_Conversational_American...

"Oh, of COURSE you will."

"Well I WILL."

"Well why don't you DO it then? What do you keep SAYING you will for? Why don't you DO it? It's because you're afraid."

"I AIN'T afraid."

"You are."

"I ain't."

"You are."

Please write 5 sentences with your opinion on what you just read.

Next Time Find out if there is anything specific the students want to learn about next week.

Extra Time Student questions, game or student directed activities.

AAR

Review what was supposed to happen.

▪

Establish what happened.

▪

Determine what was right or wrong with what happened.

▪

Determine how the task should be done differently the next time.

REVIEW

Lesson Plan

Opening Chat about our past week and answer any student questions.

Topic Review the past 9 weeks.

Goals Students write on board, "My goal today is _____."

Review Ask students what they would like to cover again, and get feedback in general on the course.

Language Talk^[4]

SENTENCE-BUILDING—Continued.

Construct sentences by supplying a predicate to each of the following subjects.

Ask yourself the question, Artists do what?

1. Artists —. 2. Sailors —. 3. Tides —. 4. Whales —. 5. Gentlemen —.
6. Swine —. 7. Clouds —. 8. Girls —. 9. Fruit —. 10. Powder —. 11.
Hail —. 12. Foxes —. 13. Water —. 14. Frost —. 15. Man —. 16.
Blood —. 17. Kings —. 18. Lilies —. 19. Roses —. 20. Wheels —. 21.
Waves —. 22. Dew —. 23. Boys —. 24. Volcanoes —. 25. Storms —.
26. Politicians —. 27. Serpents —. 28. Chimneys —. 29. Owls —. 30.
Rivers —. 31. Nations —. 32. Indians —. 33. Grain —. 34. Rogues —.
34. Volcanoes —. 35. Rome —. 36. Briars —.

This exercise may be extended by requiring the pupils to supply several predicates to each subject.

Tom Sawyer^[5]

Another pause, and more eying and sidling around each other. Presently they were shoulder to shoulder. Tom said:

"Get away from here!"

"Go away yourself!"

"I won't."

"I won't either."

So they stood, each with a foot placed at an angle as a brace, and both shoving

with might and main, and glowering at each other with hate. But neither could get an advantage. After struggling till both were hot and flushed, each relaxed his strain with watchful caution, and Tom said:

"You're a coward and a pup. I'll tell my big brother on you, and he can thrash you with his little finger, and I'll make him do it, too."

"What do I care for your big brother? I've got a brother that's bigger than he is—and what's more, he can throw him over that fence, too." [Both brothers were imaginary.]

"That's a lie."

"YOUR saying so don't make it so."

Tom drew a line in the dust with his big toe, and said:

"I dare you to step over that, and I'll lick you till you can't stand up. Anybody that'll take a dare will steal sheep."

The new boy stepped over promptly, and said:

"Now you said you'd do it, now let's see you do it."

"Don't you crowd me now; you better look out."

"Well, you SAID you'd do it—why don't you do it?"

"By jingo! for two cents I WILL do it."

The new boy took two broad coppers out of his pocket and held them out with derision. Tom struck them to the ground. In an instant both boys were rolling and tumbling in the dirt, gripped together like cats; and for the space of a minute they tugged and tore at each other's hair and clothes, punched and scratched each other's nose, and covered themselves with dust and glory. Presently the confusion took form, and through the fog of battle Tom appeared, seated astride the new boy, and pounding him with his fists. "Holler 'nuff!" said he.

The boy only struggled to free himself. He was crying—mainly from rage.

"Holler 'nuff!"—and the pounding went on.

At last the stranger got out a smothered "'Nuff!" and Tom let him up and said:

"Now that'll learn you. Better look out who you're fooling with next time."

Please write 5 sentences with your opinion on what you just read.

Extra Time Student questions, game or student directed activities.

AAR

Review what was supposed to happen.

-

Establish what happened.

-

Determine what was right or wrong with what happened.

-

Determine how the task should be done differently the next time.

NOTES

1. ↑ Note the first e-book edition was published by Mr. Danoff's Teaching Laboratory, not the Neo-Hogarth Press. The Neo-Hogarth Press was a fictitious division of the Lab that was never actually created, it was intended only as an Homage to my favorite author Virginia Woolf's famous "Hogarth Press" and had no connection whatsoever with the actual Hogarth Press. Copies of the first ed. were only available for free download online and given for free to some of my students.
2. ↑ ESL 101 Conversational American English Lesson 1 Greetings
<http://www.archive.org/details/Esl101ConversationalAmericanEnglishLesson1Greetings>
by Charles Jeffrey Danoff. Published on the Internet Archive. Public Domain.
3. ↑ http://commons.wikimedia.org/wiki/File:ESL_101_Lesson_1_Image_0.jpg (Change the last number of the URL from 0 - 14 to see all the images). All images are Public Domain.
4. ↑ 4.0 4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 Graded Lessons in English An Elementary English Grammar Consisting of One Hundred Practical Lessons, Carefully Graded and Adapted to the Class-Room. Author: Alonzo Reed and Brainerd Kellogg.
<http://archive.org/details/gradedlessonsino6kellgoog>
5. ↑ 5.0 5.1 5.2 5.3 5.4 5.5 5.6 5.7 5.8 5.9 The Adventures of Tom Sawyer, Complete by

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"

ENG 099 Conversational American English Textbook 1.1st Ed. - Paragogy.net http://paragogy.net/index.php?title=ENG_099_Conversational_American...

Mark Twain (Samuel Clemens) http://en.wikisource.org/wiki/Wikisource:Books/The_Adventures_of_Tom_Sawyer

6. ↑ <http://www.youtube.com/watch?v=sj2Yo0e8PO4>

7. ↑ About.com's Telephone Conversations: <http://esl.about.com/od/businessspeaking/a/Telephone-Conversations.htm>

8. ↑ http://www.englishclub.com/speaking/telephone_tips.htm

9. ↑ <http://www.irs.gov/formspubs/index.html>

10. ↑ <http://www.en.utexas.edu/amlit/amlitprivate/scans/goodre.html>

11. ↑ The Elements of Style by William Strunk <http://codeblab.com/elos/>

12. ↑ How to Prepare for a Job Interview : How to Answer Job Interview Questions
<http://www.youtube.com/watch?v=cCQdloL8HV0>

Retrieved from "http://paragogy.net/ENG_099_Conversational_American_English_Textbook_1.1st_Ed."

This page was last modified on 6 July 2012, at 16:10.
Content is available under CC0 1.0 Universal.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"



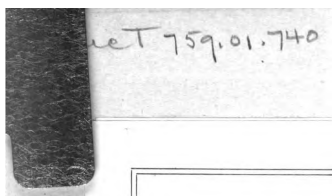
Graded lessons in English

Alonzo Reed, Brainerd Kellogg

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"



Harvard College Library

FROM

LIBRARY OF THE
Department of Education

COLLECTION OF TEXT-BOOKS

Contributed by the Publishers

TRANSFERRED
TO
HARVARD
LIBRARY



Digitized by Google

6

GRADED LESSONS IN ENGLISH.

AN ELEMENTARY ENGLISH GRAMMAR,

CONSISTING OF ONE HUNDRED PRACTICAL LESSONS CAREFULLY
GRADED AND ADAPTED TO THE CLASS-ROOM.

BY

ALONZO REED, A.M.,

FORMERLY INSTRUCTOR IN ENGLISH GRAMMAR IN THE POLYTECHNIC INSTITUTE, BROOKLYN

AND

BRAINERD KELLOGG, LL.D.,

PROFESSOR OF THE ENGLISH LANGUAGE AND LITERATURE IN THE
POLYTECHNIC INSTITUTE, BROOKLYN.

REVISED EDITION, 1901.

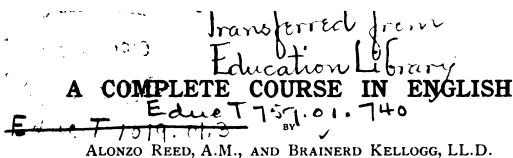
NEW YORK:

MAYNARD, MERRILL, & CO., PUBLISHERS.

1901.

Digitized by Google

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"



REED'S WORD LESSONS. A Complete Speller. Designed to teach the correct spelling, pronunciation, and use of such words only as are most common in current literature, and as are most likely to be misspelled, mispronounced, or misused, and to awaken new interest in the study of synonyms and of word-analysis. 188 pages, 12mo.

REED'S INTRODUCTORY LANGUAGE WORK. A simple, varied, and pleasing, but methodical series of exercises in English to precede the study of technical grammar. 253 pages, 16mo, cloth.

REED & KELLOGG'S GRADED LESSONS IN ENGLISH. An elementary English grammar, consisting of one hundred practical lessons, carefully graded and adapted to the class room. 280 pages, 16mo, cloth.

REED & KELLOGG'S HIGHER LESSONS IN ENGLISH. A work on English grammar and composition, in which the science of the language is made tributary to the art of expression. A course of practical lessons carefully graded, and adapted to everyday use in the schoolroom. 386 pages, 16mo, cloth.

REED & KELLOGG'S HIGH SCHOOL GRAMMAR. A work dealing with the science of the English language, the history of the parts of speech, the philosophy of the changes these have undergone, and with present usage respecting forms in dispute. 285 pages, 16mo, cloth.

KELLOGG & REED'S WORD-BUILDING. Fifty lessons, combining Latin, Greek, and Anglo-Saxon roots, prefixes, and suffixes, into about fifty-five hundred common derivative words in English; with a brief history of the English language. 122 pages, 16mo, cloth.

KELLOGG & REED'S THE ENGLISH LANGUAGE. A brief history of the grammatical changes of the language and its vocabulary, with exercises on synonyms, prefixes, suffixes, word-analysis, and word-building. A text-book for high schools and colleges. 220 pages, 16mo, cloth.

KELLOGG'S TEXT-BOOK ON RHETORIC. Revised and enlarged edition. Supplementing the development of the science with exhaustive practice in composition. A course of practical lessons adapted for use in high schools, academies, and lower classes of colleges. 345 pages, 12mo, cloth.

KELLOGG'S TEXT-BOOK ON ENGLISH LITERATURE. With copious extracts from the leading authors, English and American, and full instructions as to the method in which the book is to be studied. 485 pages, 12mo, cloth.

COPYRIGHT 1889, 1894, 1896 BY ALONZO REED AND BRAINERD KELLOGG;
AND 1901 BY FRANCES M. REED AND BRAINERD KELLOGG.

Digitized by Google

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"

1901, Dec. 9.
Harvard University,
Dept. of Education Library.
Gift of the Publishers.

PREFACE.

THE plan of "Graded and Higher Lessons in English" will perhaps be better understood if we first speak of two classes of text-books with which this course is brought into competition.

Method of One Class of Text-books.—In one class are those that aim chiefly to present a course of technical grammar in the order of Orthography, Etymology, Syntax, and Prosody. These books give large space to grammatical Etymology, and demand much memorizing of definitions, rules, declensions, and conjugations, and much formal word parsing, — work of which a considerable portion is merely the invention of grammarians, and has little value in determining the pupil's use of language or in developing his reasoning faculties. This is a revival of the long-endured, unfruitful, old-time method.

Method of Another Class of Text-books.—In another class are those that present a miscellaneous collection of lessons in Composition, Spelling, Pronunciation, Sentence-analysis, Technical Grammar, and General Information, without unity or continuity. The pupil who completes these books will have gained something by practice and will have picked up some scraps of knowledge; but his information will be vague and disconnected, and he will have missed that mental training which it is the aim of a good text-book to afford. A text-book is of value just so far as it presents a clear, logical development of its subject. It must present its science or its art as a natural growth, otherwise there is no justification of its being.

The Study of the Sentence for the Proper Use of Words.—It is the plan of this course to trace with easy steps the natural development of the sentence, to consider the leading facts first and then to descend to the details. To begin with the parts of speech is to begin with details and to disregard the higher unities, without which the details are scarcely intelligible. The part of speech to which a word belongs is determined only by its function in the sentence, and inflections simply mark the offices and relations of words. Unless the pupil has been systematically trained to discover the functions and relations of words as elements of an

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

organic whole, his knowledge of the parts of speech is of little value. It is not because he cannot conjugate the verb or decline the pronoun that he falls into such errors as "How many sounds *have* each of the vowels?" "Five years' interest *are* due." "She is older than *me*." He probably would not say "each *have*," "interest *are*," "*me* am." One thoroughly familiar with the structure of the sentence will find little trouble in using correctly the few inflectional forms in English.

The Study of the Sentence for the Laws of Discourse.—Through the study of the sentence we not only arrive at an intelligent knowledge of the parts of speech and a correct use of grammatical forms, but we discover the laws of discourse in general. In the sentence the student should find the law of unity, of continuity, of proportion, of order. All good writing consists of good sentences properly joined. Since the sentence is the foundation or unit of discourse, it is all-important that the pupil should know the sentence. He should be able to put the principal and the subordinate parts in their proper relation; he should know the exact function of every element, its relation to other elements, and its relation to the whole. He should know the sentence as the skillful engineer knows his engine, that, when there is a disorganization of parts, he may at once find the difficulty and the remedy for it.

The Study of the Sentence for the Sake of Translation.—The laws of thought being the same for all nations, the logical analysis of the sentence is the same for all languages. When a student who has acquired a knowledge of the English sentence comes to the translation of a foreign language, he finds his work greatly simplified. If in a sentence of his own language he sees only a mass of unorganized words, how much greater must be his confusion when this mass of words is in a foreign tongue! A study of the parts of speech is a far less important preparation for translation, since the declensions and conjugations in English do not conform to those of other languages. Teachers of the classics and of modern languages are beginning to appreciate these facts.

The Study of the Sentence for Discipline.—As a means of discipline nothing can compare with a training in the logical analysis of the sentence. To study thought through its outward form, the sentence, and to discover the fitness of the different parts of the expression to the parts of the thought, is to learn to think. It has been noticed that pupils thoroughly trained in the analysis and the construction of sentences come to their other studies with a decided advantage in intellectual power. These results can be obtained only by systematic and persistent work. Experienced teachers understand that a few weak lessons on the sentence

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Preface.

5

at the beginning of a course and a few at the end can afford little discipline and little knowledge that will endure, and that a knowledge of the sentence cannot be gained by memorizing complicated rules and labored forms of analysis. To compel a pupil to wade through a page or two of such bewildering terms as "complex adverbial element of the second class" and "compound prepositional adjective phrase," in order to comprehend a few simple functions, is grossly unjust; it is a substitution of form for content, of words for ideas.

Subdivisions and Modifications after the Sentence.—Teachers familiar with text-books that group all grammatical instruction around the eight parts of speech, making eight independent units, will not, in the following lessons, find everything in its accustomed place. But, when it is remembered that the thread of connection unifying this work is the sentence, it will be seen that the lessons fall into their natural order of sequence. When, through the development of the sentence, all the offices of the different parts of speech are mastered, the most natural thing is to continue the work of classification and subdivide the parts of speech. The inflection of words, being distinct from their classification, makes a separate division of the work. If the chief end of grammar were to enable one to parse, we should not here depart from long-established precedent.

Sentences in Groups—Paragraphs.—In tracing the growth of the sentence from the simplest to the most complex form, each element, as it is introduced, is illustrated by a large number of detached sentences, chosen with the utmost care as to thought and expression. These compel the pupil to confine his attention to one thing till he gets it well in hand. Paragraphs from literature are then selected and are used at intervals, with questions and suggestions to enforce principles already presented, and to prepare the way informally for the regular lessons that follow. The lessons on these selections are, however, made to take a much wider scope. They lead the pupil to discover how and why sentences are grouped into paragraphs, and how paragraphs are related to each other; and they lead him on to discover whatever is most worthy of imitation in the style of the several models presented.

The Use of the Diagram.—In written analysis, the simple map, or diagram, found in the following lessons, will enable the pupil to present directly and vividly to the eye the exact function of every clause in the sentence, of every phrase in the clause, and of every word in the phrase—to picture the complete analysis of the sentence, with principal and subordinate parts in their proper relations. It is only by the aid of such a map, or picture, that the pupil can, at a single view, see the sentence

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

as an organic whole made up of many parts performing various functions and standing in various relations. Without such a map he must labor under the disadvantage of seeing all these things by piecemeal or in succession.

But, if for any reason the teacher prefers not to use these diagrams, they may be omitted without causing the slightest break in the work. The plan of this book is in no way dependent on the use of the diagrams.

The Objections to the Diagram.—The fact that the pictorial diagram groups the parts of a sentence according to their offices and relations, and not in the order of speech, has been spoken of as a fault. It is, on the contrary, a merit, for it teaches the pupil to look through the literary order and discover the logical order. He thus learns what the literary order really is, and sees that this may be varied indefinitely, so long as the logical relations are kept clear.

The assertion that correct diagrams can be made mechanically is not borne out by the facts. It is easier to avoid precision in oral analysis than in written. The diagram drives the pupil to a most searching examination of the sentence, brings him face to face with every difficulty, and compels a decision on every point.

The Abuse of the Diagram.—Analysis by diagram often becomes so interesting and so helpful that, like other good things, it is liable to be overdone. There is danger of requiring too much written analysis. When the ordinary constructions have been made clear, diagrams should be used only for the more difficult sentences; or, if the sentences are long, only for the more difficult parts of them. In both oral and written analysis there is danger of repeating what needs no repetition. When the diagram has served its purpose, it should be dropped.

TABLE OF CONTENTS.

TWENTY-FIVE CAREFULLY GRADED STEPS IN GRAMMAR AND COMPOSITION;
AND RULES FOR CAPITALIZATION AND PUNCTUATION
AS NEEDED.

	PAGE
1. Analysis and Composition of Sentences with Simple Subjects and Predicates — Capital Letters, Period, Interrogation Point, Parts of Speech, Nouns, Verbs, Pronouns	9-33
2. Analysis and Composition of Sentences with Subjects modified by Adjectives	33-44
3. Analysis and Composition of Sentences with Predicates modified by Adverbs	44-60
4. Analysis and Composition of Sentences with Subjects and Predicates modified by Prepositional Phrases—The Paragraph, Prepositions	60-66
5. Expansion of Adjectives and Adverbs into Phrases, and Contraction of Phrases into Adjectives and Adverbs	66, 67
6. Analysis and Composition of Sentences with Compound Subjects and Predicates — Conjunctions, Interjections, Exclamation Point	70-79
7. Analysis and Composition of Sentences with Nouns, Pronouns, and Adjectives as Complements—Object Complement, Attribute Complement, Narration, Position and Use of Modifiers	79-84, 95-97
8. Analysis and Composition of Sentences with Participles and Infinitive Phrases—The Participle, Descriptive Writing	102-108
9. Analysis and Composition of Sentences with Nouns and Pronouns as Modifiers—Comma, Argument	115-120
10. Analysis and Composition of Complex Sentences containing Adjective Clauses	124-128
11. Analysis and Composition of Complex Sentences containing Adverb Clauses	129-132

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

	PAGE
12. Analysis and Composition of Complex Sentences containing Noun Clauses	136-139
13. Analysis and Composition of Compound Sentences — Independent Clauses	140, 141
14. Declarative, Interrogative, Imperative, and Exclamatory Sentences	141-144
15. Expansion and Contraction — Continued	147-150
16. Classes of Nouns and Pronouns in Sentences	157-160
17. Classes of Adjectives in Sentences	161, 162
18. Classes of Verbs in Sentences	162-165
19. Classes of Adverbs in Sentences	166, 167
20. Classes of Conjunctions and of Other Connectives in Sentences	167-170
21. Nouns and Pronouns with all their Modifications in Sentences	176-192
22. Adjectives and Adverbs with their one Modification in Sentences	193-197
23. Verbs with all their Modifications in Sentences	197-217
24. Composition of Sentences in Paragraphs and of Paragraphs in Themes	56-60, 86-89, 108-113, 122-124, 133-136, 145, 146, 152-156, 171-174
25. Composition of Paragraphs in Letters — Summary of Rules of Syntax, Proof Marks	220-246
REVIEW OF GRADED LESSONS	247-271
ABBREVIATIONS	272
INDEX	279

A TALK ON LANGUAGE.

THE teacher is recommended to occupy the time of at least two or three recitations, in talking with his pupils about language, always remembering that, in order to secure the interest of his class, he must allow his pupils to take an active part in the exercise. The teacher should guide the thought of his class; but, if he attempts to do all the talking, he will find, when he concludes, that he has been left to do all the thinking.

We give below a few hints in conducting this talk on language, but the teacher is not expected to confine himself to them. He will, of course, be compelled, in some instances, to resort to various devices in order to obtain from the pupils answers equivalent to those here suggested.

LESSON 1.

Teacher. — I will pronounce these three sounds very slowly and distinctly, thus: *b-u-d*. Notice, it is the power, or sound, of the letter, and not its name, that I give. What did you hear?

Pupil. — I heard three sounds.

T. — Give them. I will write on the board, so that you can see them. three letters — *b-u-d*. Are these letters, taken separately, signs to you of anything?

P. — Yes, they are signs to me of the three sounds that I have just heard.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

10

Graded Lessons In English.

T. — What then do these letters, taken separately, picture to your eye?

P. — They picture the sounds that came to my ear.

T. — Letters then are the signs of what?

P. — **Letters are the signs of sounds.**

T. — I will pronounce the same sounds rapidly, uniting them more closely — *bud*. These sounds, so united, form a spoken word. Of what do you think when you hear the word *bud*?

P. — I think of a little round thing that grows to be a leafy branch or a flower.

T. — Did you see the thing when you were thinking of it?

P. — No.

T. — Then you must have had a picture of it in your mind. We call this **mental picture** an **idea**. What called up this idea?

P. — It was called up by the word *bud*, which I heard.

T. — A spoken word then is the sign of what?

P. — **A spoken word is the sign of an idea.**

T. — I will call up the same idea in another way. I will write three letters and unite them thus: *bud*. What do you see?

P. — I see the word *bud*.

T. — If we call the other word *bud* a spoken word, what shall we call this?

P. — This is a written word.

T. — If they stand for the same idea, how do they differ?

P. — I see this, and I heard that.

T. — You will observe that we have called attention to four different things; viz., the **real bud**; your mental picture of the bud, which we have called an **idea**; and the **two words**, which we have called signs of this idea, the one addressed to the ear, and the other to the eye.

If the pupil be brought to see these distinctions, it may aid him to observe more closely and express himself more clearly.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

A Talk on Language.

11

LESSON 2.

Teacher. — What did you learn in the previous Lesson?

Pupil. — I learned that a spoken word is composed of certain sounds; that a written word is composed of letters; that letters are signs of sounds; and that spoken and written words are the signs of ideas.

This question should be passed from one pupil to another till all of these answers are elicited.

All the written words in all the English books ever made are formed of twenty-six letters, representing about forty-four sounds. These letters and these sounds make up what is called verbal language.

Of these twenty-six letters, **a, e, i, o, u,** and sometimes **w** and **y,** are called **vowels**, and the remainder are called **consonants**.

In order that you may understand what kind of sounds the vowels stand for, and what kinds the consonants represent, I will tell you something about the human voice.

The air breathed out from your lungs beats against two flat muscles, stretched like strings across the top of the windpipe, and causes them to vibrate. This vibration makes sound. Put one end of a thread between your teeth, hold the other end of it in your fingers, draw it tight and strike it, and you will understand how voice is made. If the voice thus produced comes out through the open mouth, a class of sounds is formed which we call vowel sounds.

But, if the voice is held back by your palate, tongue, teeth, or lips, one kind of consonant sounds¹ is made. If the breath is driven out without voice, and is held back by these same parts of the mouth, the other kind of consonant sounds¹ is formed.

¹ Called respectively **sonants** and **surds**. We suggest that you have the pupils give the sounds of the vowel **a** in *ale, care, am, arm, ask,* and

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

12

Graded Lessons in English.

The teacher and pupils should practice on these sounds till the three kinds can easily be distinguished.

You are now prepared to understand what I mean when I say that the **vowels** are the **letters** which stand for the **open sounds of the voice**, and that the **consonants** are the **letters** which stand for the sounds made by the **obstructed voice** and the **obstructed breath**.

The teacher can here profitably spend a few minutes in showing how ideas may be communicated by Natural Language, the language of sighs, groans, gestures of the hands, attitudes of the body, expressions of the face, tones of the voice, etc. He can show that, in conversation, we sometimes couple this Natural Language of tone and gesture with our language of words in order to make a stronger impression. Let the pupil be told that, if the passage contain feeling, he should do the same in Reading and Declaiming.

Let the following definitions be learned, and given at the next recitation.

DEFINITION. — Verbal language, or **language proper**, consists of the spoken and written words used to communicate ideas and thoughts.

DEFINITION. — **English grammar** is the science which teaches the forms, uses, and relations of the words of the English language.

all; the sounds of **e** in *ye*, *end*, and *fern*; of **i** in *ice* and *ill*; of **o** in *old*, *orb*, and *odd*; and of **u** in *use*, *rude*, *full*, *up*, and *urn*. The sounds of the sonants **b**, **d**, **g** in *gin*, **g** in *get*, **j**, **l**, **m**, **n**, **r**, **s** in *is*, **s** in *vision*, **v**, **x** in *Xenophon*, **x** in *exact*, **z** in *zero*, and **z** in *seizure*; and the sounds of the surds, **c** in *cent*, **c** in *cat*, **f**, **h**, **k**, **p**, **q**, **s**, **s** in *sure*, **t**, and **x** in *wax*.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

A Talk on Language.

13

LESSON 3.

Let the pupils be required to tell what they learned in the previous Lessons.

Teacher. — When I pronounce the two words *star* and *bud*, thus : *star bud*, how many ideas, or mental pictures, do I call up to you?

Pupil. — Two.

T. — Do you see any connection between these ideas?

P. — No.

T. — When I utter the two words *bud* and *swelling* thus : *bud swelling*, do you see any connection in the ideas they stand for?

P. — Yes, I imagine that I see a bud expanding, or growing larger.

T. — I will connect two words more closely, so as to express a thought: "*Buds swell.*" A thought has been formed in my mind when I say, "*Buds swell*"; and these two words, by which something is said of something else, express that thought, and make what we call a sentence. In the former expression, *bud swelling*, it is assumed, or taken for granted, that buds perform the act; in the latter, the swelling is asserted as a fact.

Leaves falling. Do these two words express two ideas merely associated, or do they express a thought?

P. — They express ideas merely associated.

T. — "*Leaves fall.*" What do these two words express?

P. — A thought.

T. — Why?

P. — Because, in these words, there is something said or asserted of leaves.

T. — When I say, "*Falling leaves rustle,*" does *falling* tell what is thought of leaves?

P. — No.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

14

Graded Lessons In English.

T. — What does *falling* do?

P. — It tells the kind of leaves you are thinking and speaking of.

T. — What word does tell what is thought of leaves?

P. — *Rustle*.

T. — You see then that in the thought there are two parts; something of which we think, and that which we think about it.

Let the pupils give other examples.

LESSON 4.

Commit to memory all definitions.

DEFINITION. — A **Sentence**¹ is the expression of a thought in words.

Which of the following expressions contain words that have no connection, which contain words merely associated, and which are sentences?

- | | | |
|---------------------|------------------------|-------------------------|
| 1. Flowers bloom. | 12. Sugar graze. | 23. Birds chirp. |
| 2. Ice melts. | 13. Dew sparkles. | 24. Gentle cows. |
| 3. Bloom ice. | 14. Hissing serpents. | 25. Eagles are soaring. |
| 4. Grass grows. | 15. Smoke curls. | 26. Bees ice. |
| 5. Brooks babble. | 16. Serpents hiss. | 27. Working bees. |
| 6. Babbling brooks. | 17. Smoke curling. | 28. Bees work. |
| 7. Grass soar. | 18. Serpents sparkles. | 29. Crawling serpents. |
| 8. Doors open. | 19. Melting babble. | 30. Landscape piano. |
| 9. Open doors. | 20. Eagles soar. | 31. Serpents crawl. |
| 10. Cows graze. | 21. Birds chirping. | 32. Eagles clock. |
| 11. Curling smoke. | 22. Bird are chirping. | 33. Serpents crawling. |

¹ Or, if preferred, A Sentence is a group of words expressing a thought.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Analysis.

15

LESSON 5.

REVIEW QUESTIONS.

Illustrate, by the use of **a**, **b**, and **p**, the difference between the sounds of letters and their names. Letters are the signs of what? What is an idea? A spoken word is the sign of what? A written word is the sign of what? How do they differ? To what four different things did we call attention in Lesson 1?

How is voice made? How are vowel sounds made? How are the two kinds of consonant sounds made? What are vowels? Name them. What are consonants? What is language proper? What do you understand by natural language? What is English grammar?

What three kinds of expressions are spoken of in Lessons 3 and 4? Give examples of each. What is a sentence?

LESSON 6.

ANALYSIS.

On the following sentences, let the pupils be exercised according to the model.

Model. — *Intemperance degrades.* Why is this a sentence?
Ans. — It expresses a thought. Of what is something thought?
Ans. — Intemperance. Which word tells what is thought? **Ans.** — *Degrades.*

- | | | |
|---------------------|---------------------|----------------------|
| 1. Magnets attract. | 5. Sunbeams dance. | 9. Grass withers. |
| 2. Horses neigh. | 6. Heat expands. | 10. Sailors climb. |
| 3. Frogs leap. | 7. Sunlight gleams. | 11. Rabbits burrow. |
| 4. Cold contracts. | 8. Banners wave. | 12. Spring advances. |

You see that in these sentences there are two parts. The parts are the **Subject** and the **Predicate**.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

16

Graded Lessons in English.

DEFINITION.—The **Subject of a sentence** names that of which something is thought.

DEFINITION.—The **Predicate of a sentence** tells what is thought.

DEFINITION.—The **Analysis of a sentence** is the separation of it into its parts.

Analyze, according to the model, the following sentences.

Oral Analysis.—*Stars twinkle.* This is a sentence, because it expresses a thought. *Stars* is the subject, because it names that of which something is thought; *twinkle* is the predicate, because it tells what is thought.

To the Teacher.—After the pupils become familiar with the definitions, the "Models" may be varied, and some of the reasons may be made specific; as, "*Plants* names the things we tell about; *droop* tells what plants do," etc.

Guard against needless repetition.

- | | | |
|--------------------------|---------------------|---------------------|
| 1. Plants droop. | 5. Rain falls. | 9. Boats capsize. |
| 2. Books help. | 6. Time flies. | 10. Water flows. |
| 3. Clouds float. | 7. Rowdies fight. | 11. Students learn. |
| 4. Exercise strengthens. | 8. Bread nourishes. | 12. Horses gallop. |

LESSON 7.

ANALYSIS AND THE DIAGRAM.

Hints for Oral Instruction.—I will draw on the board a heavy, or shaded, line, and divide it into two parts, thus:



Digitized by Google

We will consider the first part as the sign of the subject of a sentence, and the second part as the sign of the predicate of a sentence.

Now, if I write a word over the first line, thus—you will understand that that word is the subject of a sentence. If I write a word over the second line, thus—you will understand that that word is the predicate of a sentence.

<u>Planets</u>		<u>revolve.</u>
----------------	--	-----------------

The class can see by this that "*Planets revolve*" is a sentence, that *planets* is the subject, and that *revolve* is the predicate.

Such pictures, made up of straight lines, we call **Diagrams**.

DEFINITION.—A **Diagram** is a picture of the offices and relations of the different parts of a sentence.

Analyze and diagram the following sentences:—

- | | | |
|---------------------|--------------------|-----------------------------|
| 1. Waves dash. | 9. Nero fiddled. | 17. Morning dawns. |
| 2. Kings reign. | 10. Larks sing. | 18. Showers descended. |
| 3. Fruit ripens. | 11. Water ripples. | 19. Diamonds sparkle. |
| 4. Stars shine. | 12. Lambs frisk. | 20. Alexander conquered. |
| 5. Steel tarnishes. | 13. Lions roar. | 21. Jupiter thunders. |
| 6. Insects buzz. | 14. Tigers growl. | 22. Columbus sailed. |
| 7. Paul preached. | 15. Breezes sigh. | 23. Grammarians differ. |
| 8. Poets sing. | 16. Carthage fell. | 24. Cornwallis surrendered. |

In Lessons 6 and 7, you notice (1) that such subjects as *time*, *Nero*, and *morning*, each denoting only one person or

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

18

Graded Lessons in English.

thing, do not add the **s**-ending ; and (2) that the predicates of such subjects do add it. Such subjects as *books*, *kings*, and *lions*, (3) each denoting more than one, add the **s**-ending ; and (4) the predicates of such subjects do not add it.

This use of the simple form of the predicate with the **s**-form of the subject, and of the **s**-form of the predicate with the simple form of the subject, is called the **agreement** of the predicate with its subject.

Note, however, that, as in 7, 16, and 22 above, the **s**-form of the predicate is not used in telling what a person or thing did—only in telling what it does now.

LESSON 8.

COMPOSITION.

You have now learned to analyze sentences, that is, to separate them into their parts. You must next learn to put these parts together, that is, to build sentences. If the separation of a sentence into its parts is **analysis**, the putting of its parts together is **synthesis**, **construction**, or **composition**.

We will find one part, and you must find the other and do the building.

To the Teacher.—Let some of the pupils write their sentences on the board while others are reading theirs. Then let the work on the board be corrected.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Composition.

19

Correct any expression that does not make good sense, or that asserts something not strictly true; for the pupil should early be taught to think accurately, as well as to write and speak grammatically.

Correct all mistakes in spelling, and in the use of capital letters and the period.

Insist on neatness. Collect the papers before the recitation closes.

CAPITAL LETTER—RULE.—The first word of every sentence must begin with a **capital letter**.

PERIOD—RULE.—A **period** must be placed after every sentence that simply affirms, denies, or expresses a command.

Construct sentences by supplying a subject to each of the following predicates:—

Ask yourself the question, What swim, sink, hunt, etc. ?
The proper answers will be the subjects required.

- | | | | |
|--------------|--------------|------------------|-------------------|
| 1. — swim. | 7. — climb. | 13. — flashes. | 19. — expand. |
| 2. — sinks. | 8. — creep. | 14. — flutters. | 20. — jump. |
| 3. — hunt. | 9. — run. | 15. — paddle. | 21. — hop. |
| 4. — skate. | 10. — walk. | 16. — toil. | 22. — bellow. |
| 5. — jingle. | 11. — snort. | 17. — terrifies. | 23. — burns. |
| 6. — decay. | 12. — kick. | 18. — rages. | 24. — evaporates. |

This exercise may profitably be extended by requiring the pupils to supply several subjects to each predicate.

Add the **s**-ending to the predicates that are without it, and make the needed change in your subjects; drop the **s**-ending from the predicates that have it, and make the needed change in your subjects.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

20

Graded Lessons in English.

LESSON 9.

COMPOSITION — *Continued.*

Construct sentences by supplying a predicate to each of the following subjects: —

Ask yourself the question, Artists, sailors, tides, etc. do what? The proper answers will be the predicates required:

- | | | |
|-----------------|------------------|--------------------|
| 1. Artists —. | 13. Water —. | 25. Storms —. |
| 2. Sailors —. | 14. Frost —. | 26. Politicians —. |
| 3. Tides —. | 15. Man —. | 27. Serpents —. |
| 4. Whales —. | 16. Blood —. | 28. Chimneys —. |
| 5. Gentlemen —. | 17. Kings —. | 29. Owls —. |
| 6. Swine —. | 18. Lilies —. | 30. Rivers —. |
| 7. Clouds —. | 19. Roses —. | 31. Nations —. |
| 8. Girls —. | 20. Wheels —. | 32. Indians —. |
| 9. Fruit —. | 21. Waves —. | 33. Grain —. |
| 10. Powder —. | 22. Dew —. | 34. Rogues —. |
| 11. Hail —. | 23. Boys —. | 35. Rome —. |
| 12. Foxes —. | 24. Volcanoes —. | 36. Briers —. |

This exercise may be extended by requiring the pupils to supply several predicates to each subject.

Add or drop the **s**-ending, and make the needed change in the predicates that they may agree.

You cannot become too familiar with the agreement of predicate with subject, and you cannot become familiar with it too early.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Analysis.

21

LESSON 10.

REVIEW QUESTIONS.

Of what two parts does a sentence consist? What is the subject of a sentence? What is the predicate of a sentence? What is the analysis of a sentence? What is synthesis, or composition?

What is a diagram? What rule for the use of capital letters have you learned? What rule for the period?

IMPROMPTU EXERCISE.

Let the pupils "choose sides," as in a spelling match. Let the teacher select predicates from Lesson 8, and give them alternately to the pupils thus arranged. The first pupil prefixes to his word whatever suitable subjects he can think of, the teacher judging of their fitness and keeping the count. This pupil now rises and remains standing until some one else, on his side or the other, shall have prefixed to his word a greater number of apt subjects. The struggle is to see who shall be standing at the close of the match, and which side shall have furnished the greater number of subjects. The exercise may be continued with the subjects of Lesson 9. The pupils are limited to the same time — one or two minutes.

LESSON 11.

ANALYSIS.

The **predicate** sometimes contains **more than one word**.

Analyze and diagram according to the model: —

Model. — *Socrates was poisoned.*

Socrates | was poisoned

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

22

Graded Lessons In English.

Oral Analysis.—This is a sentence, because it expresses a thought. *Socrates* is the subject, because — ; *was poisoned* is the predicate, because ¹ —.

- | | |
|-------------------------------|--|
| 1. Napoleon was banished. | 14. Snow is falling. |
| 2. André was captured. | 15. Leaves are rustling. |
| 3. Money is circulated. | 16. Children will prattle. |
| 4. Columbus was imprisoned. | 17. Crickets are chirping. |
| 5. Acorns are sprouting. | 18. Eclipses have been foretold. |
| 6. Bells are tolled. | 19. Storms may abate. |
| 7. Summer has come. | 20. Deception may have been practiced. |
| 8. Sentences may be analyzed. | 21. Esau was hated. |
| 9. Clouds are reddening. | 22. Treason should have been punished. |
| 10. Air may be weighed. | 23. Bees are humming. |
| 11. Jehovah shall reign. | 24. Sodom might have been spared. |
| 12. Corn is planted. | |
| 13. Grammarians will differ. | |

Notice that *is*, *was*, *has*, and also *does*, are used with subjects denoting but one ; and that *are*, *were*, *have*, and also *do*, are used with subjects denoting more than one.

Drop the *s*-ending from the subjects of 5, 6, 9, 15, 17, 18, and 23, and change *are* and *have*, that predicate and subject may agree.

Exchange the subjects of 1, 2, 3, 4, 7, 12, 14, and 21, for others with *s*-ending, and change *was*, *is*, and *has*, that predicates may agree with subjects.

¹ The word *because*—suggesting a reason—should be dropped from these “Models” whenever it may lead to mere mechanical repetition. Avoid deadly routine at whatever cost.

LESSON 12.

COMPOSITION.

Prefix the little helping words in the second column to such of the more important words in the third column as with them will make complete predicates, and join these predicates to all subjects in the first column with which they will unite and make good sense.

1	2	3
Burgoyne	are	woven.
Henry Hudson	was	defeated.
Sparrows	can be	condensed.
Comets	is	inhaled.
Time	have been	worn.
Turbans	may be	slacked.
Lime	has been	wasted.
Steam	could have been	seen.
Air	must have been	deceived.
Carpets	were	quarreling.

LESSON 13.

Point out the subject and the predicate of each sentence in Lessons 28, 31, 34.

Look first for the word that asserts, and then, by putting *who* or *what* before this predicate, the subject may easily be found.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

24

Graded Lessons in English.

Read aloud in the class the sentences of Lesson 11 with the helping words *is, was, may, are, should*, etc. before their subjects. Read also the first nine sentences of Lesson 31 with *does* or *did* — as the case requires — before the subjects, making the needed changes in the predicates.

The sentences thus read become **interrogative**, ask questions; and, if written, the **interrogation point** would be used. See Lesson 63.

To the Teacher. — Most violations of the rules of agreement come from a failure to recognize the relation of subject and predicate when these parts are transposed or are separated by other words. Such constructions should therefore receive special attention.

Introduce the class to the Parts of Speech before the close of this recitation. See "Hints for Oral Instruction."

LESSON 14.

CLASSES OF WORDS.

Hints for Oral Instruction. — By the assistance of the few hints here given, the ingenious teacher may render this usually dry subject interesting and attractive. By questioning the pupil as to what he has seen and heard, his interest may be excited and his curiosity awakened.

Suppose that we make an imaginary excursion to some field or grove, where we may study the habits, the plumage, and the songs of the birds.

If we attempt to make the acquaintance of every little

Digitized by Google

feathered singer we meet, we shall never get to the end of our pleasant task ; but we find that some resemble one another in size, shape, color, habits, and song. We associate these together and call them sparrows.

We find others differing essentially from the sparrows, but resembling one another. These we call robins. Others, for like reasons, we call bobolinks.

We thus find that, although we cannot become acquainted with each individual bird, they all belong to a few classes, with which we may soon become familiar.

It is so with the words of our language. There are many thousands of them, and they all belong to **eight classes**, called **Parts of Speech**.

We classify birds according to their form, color, etc., but we group words into classes, called **Parts of Speech**, with respect to their use in the sentence.

We find that many words are names. These we put into one class and call them **Nouns**.

Each pupil may give the name of something in the room ; the name of a distinguished person ; a name that may be applied to a class of persons ; the name of an animal ; the name of a place ; the name of a river ; the name of a mountain ; the name of something which we cannot see or touch, but of which we can think ; as, *beauty*, *mind*.

Remind the pupils frequently that these names are all nouns.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

26

Graded Lessons in English.

NOUNS.

DEFINITION.—A **Noun** is the name of anything.

Write in columns, headed nouns, the names of domestic animals, of garden vegetables, of flowers, of trees, of articles sold in a dry-goods store, and of things that cannot be seen or touched; as, *virtue, time, life*.

Write and arrange, according to the following model, the names of things that can float, fly, walk, work, sit, or sing:—

<i>Nouns.</i>	
Cork	} floats or float.
Clouds	
Model. —Wood	
Ships	
Boys	

Such expressions as "*Cork floats*" are sentences, and the nouns *cork, ships*, etc. are the subjects. You will find that **every subject** is a **noun** or some word or words used for a noun.

Be prepared to analyze and parse the sentences which you have made. Naming the class to which a word belongs is the **first step** in **parsing**.

Oral Analysis.—This is a sentence, because —; *cork* is the subject, because —; *floats* is the predicate, because —.

Parsing.—*Cork* is a noun, because it is the name of a thing—the bark of a tree.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Verbs.

27

LESSON 15.

Select and write all the nouns in the sentences given in Lessons 28, 31, 34.

Tell why they are nouns.

In writing the nouns, observe the following rule:—

CAPITAL LETTER—RULE.—Every **proper** or individual **name** must begin with a **capital** letter.

REVIEW QUESTIONS.

With respect to what do we classify words (Lesson 14)? What are such classes called? Can you illustrate this classification? What are all names? What is a noun? What is the first step in parsing? What is the rule for writing proper names?

LESSON 16.

VERBS.

Hints for Oral Instruction.—We introduce you now to another class of words. You have learned that one very large class consists of names of things. There is another very important class used to tell what these things do, or used to express their existence.

When I say, "Plants *grow*," is *grow* the name of anything? P.—No. T.—What does it do? P.—It tells what plants do. It expresses action.

T.—When I say, "God *is*," what does *is* express? P.—It expresses existence, or being.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

28

Graded Lessons in English.

T.—When I say, "George *sleeps*," *sleeps* expresses being and something more; it tells the condition, or state, in which George is, or exists, that is, it expresses state of being.

All the words that assert action, being, or state of being we call **Verbs**.

Let the teacher write nouns on the board, and require the pupils to give all the words of which they can think, telling what the things named can do. They may be arranged thus:—

Noun. Verbs.

Plants	{	grow,	Each pupil may give a verb that expresses an action of the body ; as, <i>weep, sing</i> ; an action of the mind ; as, <i>study, love</i> ; one that expresses being or state of being.
		droop,	
		decay,	
		flourish,	
		revive.	

DEFINITION.—A **Verb** is a word that asserts action, being, or state of being.

The office of the verb in all its forms except two (the participle and the infinitive, see Lessons 48 and 49) is to **assert**. This it does whether the sentence affirms, denies, or asks a question.

To the Teacher.—In the exercises of this and the next two Lessons, let the pupils note the agreement of the verb with its subject.

Supply to each of the following nouns as many appropriate verbs as you can think of. Let some express being or state of being:—

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Verbs.

29

Water —.	Wind —.	Pens —.	Parrots —.
Vines —.	Farmers —.	Trees —.	Ministers —.

One verb may consist of two, three, or four words; as, *is singing, will be sung, might have been sung.*

Form verbs by combining the words in columns 2 and 3, and add these verbs to all the nouns in column 1 with which they appropriately combine:—

1	2	3
Laws	has been	published.
Clouds	have been	paid.
Food	will be	restored.
Health	should have been	preserved.
Taxes	may be	collected.
Books	are	obeyed.

The examples you have written are sentences; the nouns are subjects, and the verbs are predicates.

As verbs are the only words that assert, **every predicate** must be a **verb** or must contain a verb.

Analyze and parse five of the sentences you have written.

Model.—*Laws are obeyed.* Diagram and analyze as in Lesson 11.

Parsing.—*Laws* is a noun, because —; *are obeyed* is a verb, because it asserts action.

LESSON 17.

Select and write all the verbs in the sentences given in Lessons 28, 31, 34, and tell why they are verbs.

Digitized by Google

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"

30

Graded Lessons in English.

LESSON 18.

COMPOSITION.

Out of the following nouns and verbs, build as many sentences as possible, taking care that every one makes good sense and expresses a truth:—

Poems, was conquered, lambs, rebellion, stars, forests, shone, were seen, were written, treason, patriots, meteors, fought, were discovered, frisk, Cain, have fallen, fled, stream, have crumbled, day, ages, deer, are flickering, are bounding, gleamed, voices, lamps, rays, were heard, are gathering, time, death, friends, is coming, will come.

LESSON 19.

PRONOUNS.

Hints for Oral Instruction.—We propose to introduce you now to the third part of speech. T.—If I should ask who whispered, and some boy should promptly confess, what would he say? P.—“*I* whispered.” T.—Would he mention his own name? P.—No. T.—What word would he use instead? P.—*I*.

T.—Suppose that I had spoken to that boy and had accused him of whispering, how should I have addressed him without mentioning his name? P.—“*You* whispered.” T.—What word would be used instead of the name of the boy to whom I spoke? P.—*You*.

T.—Suppose that, without using his name, I had told

Digitized by Google

you what he did, what should I have said? P.—“*He* whispered.” T.—What word would have been used instead of the name of the boy of whom I spoke? P.—“*He*.”

Repeat these questions, supposing the pupil to be a girl.

T.—If I should tell that boy to close his book when his book was already closed, what would he say without mentioning the word *book*? P.—“*It* is closed.”

T.—If I should accuse several of you of whispering, and one should speak for himself and for those whispering with him, what would he say? P.—“*We* whispered.”

T.—Suppose that a boy should inform me that all of the boys on that seat had whispered, what would he say? P.—“*They* whispered.”

I, you, he, she, it, we, and they are not names, but they are used instead of names. We call such words **Pronouns**.

DEFINITION. — A **Pronoun** is a word used for a noun.

CAPITAL LETTERS — RULE. — The words **I** and **O** should be written in capital letters.

ANALYSIS AND PARSING.

Model. — *You will be rewarded.*

Oral Analysis. — This is a sentence, because — ; *you* is the subject, because — ; *will be rewarded* is the predicate, because — .

Parsing. — *You* is a pronoun, because it stands for the name of the person spoken to ; *will be rewarded* is a verb, because — .

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Analyze these sentences, and parse the words :—

- | | |
|------------------------|-------------------------|
| 1. We think. | 6. It has been decided. |
| 2. She prattles. | 7. He was punished. |
| 3. We have recited. | 8. They are conquered. |
| 4. I study. | 9. Thou art adored. |
| 5. You have been seen. | |

You see that, without changing the verb-form, *I*, *you*, and *they* may take the place of *we* in 1 and 3 above; that *you*, *we*, and *they* may take the place of *I* in 4; that *I*, *we*, and *they* may take the place of *you* in 5; and that *I*, *we*, and *you* may take the place of *they* in such a sentence as "They *have* or *had* conquered." In other words, the pronouns *I*, *we*, *you*, and *they* require the same verb-form.

An exception is *I* with *are* and *were*—forms of the verb *be*. We may say, "*We*, *you*, and *they are* or *were* conquered"; but, using *I*, we must say, "*I am* or *was* conquered."

Thou, as in 9, is rare; *you* takes its place. *You* may mean one or more than one, but the verb always agrees with it as if it meant more than one.

He, *she*, and *it* require *is* and *was* and the s-form of the verb seen above in *has* and *prattles*. *I* cannot be the subject of *is* or of an s-form.

To the Teacher.—Over and again, till the ear is accustomed to the right sound, have your pupils repeat aloud *I* with the agreeing verb-forms *am* and *was*; *we*, *you*, and *they* with *are* and *were*; *I*, *we*, *you*, and *they* with the simple verb-forms *have*, *go*, *think*, *study*, *come*, etc.; and *he*, *she*, and *it* with *is*, *was*, and the s-forms *has*, *thinks*, *goes*,

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Modified Subject.

33

studies, etc. Guard the pupils especially against the common errors of *was* for *were*, and *don't* for *doesn't*.

Compose nine similar sentences, using a pronoun for the subject of each, and diagram them.

To the Teacher.—Before this recitation closes, explain "Modified Subject." See "Hints for Oral Instruction."

LESSON 20.

MODIFIED SUBJECT.

Hints for Oral Instruction.—You have already learned that a noun or pronoun and a verb sometimes make a complete sentence; but we are about to show you that they are often used as the foundation only of a sentence, which is completed by adding other parts.

I hold in my hand several pieces of metal, with letters and other characters stamped on them. What do you say I have in my hand? **P.**—Money. **T.**—Yes. What other word can you use? **P.**—*Coin*. **T.**—Yes. I will write on the board this sentence: "*Coin* is stamped."

Coin is a general or class name for all such pieces of metal. I will write the word *the* before this sentence: "*The coin* is stamped." I have now made an assertion about one particular coin, so the meaning of the subject is limited by joining the word *the*.

I can limit the meaning of the subject by putting the word *a* before it. The assertion is now about one coin,

Digitized by Google

but no particular one. I point to the piece near me and say, "*This coin* is stamped." I point to the one farther from me and say, "*That coin* is stamped."

When words are joined to the subject to limit its meaning, we say that the subject is modified.

The words *the*, *a*, *this*, and *that* modify the subject by limiting the word to one coin, or to one particular coin.

We can modify the subject by joining some word which will tell what kind of coin is meant.

Here is a coin dated 19—. We can say, "*The new coin* is stamped." Here the word *new* tells what kind of coin is meant. What other words can I use to modify *coin*?
P. — *Beautiful, bright, round, silver.* **T.** — These words *beautiful, bright, round,* and *silver* modify the subject by telling the qualities of the coin.

We call the words *the, beautiful,* etc. **Modifiers.**

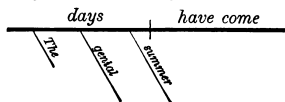
DEFINITION. — A **Modifier** is a word or group of words joined to some part of the sentence to qualify or limit the meaning.

The **Subject** with its **Modifiers** is called the **Modified Subject.**

ANALYSIS.

Analyze and diagram the following sentences:—

Model. — *The genial summer days have come.*



The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Modified Subject.

35

Explanation of the Diagram. — The lighter lines, joined to the subject line, stand for the modifiers, the less important parts.

Oral Analysis. — This is a sentence, because — ; *days* is the subject, because — ; *have come* is the predicate, because — ; *The, genial, and summer* are modifiers of the subject, because they are words joined to the subject to modify its meaning. *The genial summer days* is the modified subject.

To the Teacher. — To excite thought and guard against mere routine, pupils may, so far as they are able, make the reasons specific. For example, "*The* points out some particular clouds, *dark* tells their color," etc.

Here and elsewhere the teacher must determine how far it is profitable to follow "Models." There is great danger of wasting time in repeating forms that require no mental effort.

1. The angry wind is howling.
2. The dead leaves fall.
3. The dark clouds lower.
4. The tall elm bends.
5. All men must die.
6. The lusty bellows roared.
7. A boding silence reigned.
8. Little Arthur was murdered.
9. The mighty oak was uprooted.
10. The fragile violet was crushed.
11. The beautiful marble statue was carved.
12. The turbid torrent roared.
13. The affrighted shepherds fled.
14. The vivid lightning flashes.
15. Those elegant Etruscan vases are broken.

Change the place of certain words in 1, 5, 8, 9, 10, 11, and 15, and read these sentences as questions — see Lesson

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

36

Graded Lessons in English.

13. Select from *do*, *did*, and *does* — forms of *do* — and read 4 and 14 as questions; do the same with 2 and 3; and with 6, 7, 12, and 13.

REVIEW QUESTIONS.

What is a verb? Give examples of verbs of action. Of being. Of state of being. Of how many words may a verb consist? Illustrate. Verbs are the only words that do what? What must every predicate contain?

What parts of speech are explained in the preceding Lessons? Give the definition of a pronoun. Give the rule for writing the words *I* and *O*.

Which one of these forms of the verb *be* — *am*, *art*, *is*, *was*, *are*, and *were* — is used with *I* only? Which with *I*, *he*, *she*, and *it*? In assertions of fact, like those thus far seen, what four forms of *be* is *I* not the subject of? (*Were* in certain uses, as we shall hereafter see, may be used with *I*, *he*, *she*, *it*, and nouns naming but one.) What forms of *be* can *he*, *she*, *it*, and nouns naming but one never be the subject of? Only what forms of *be* may *we*, *you*, *they*, and nouns naming more than one be the subject of? Which only of these forms of *have* — *has*, *hast*, *have*, and *had* — may *I* be the subject of? *He*, *she*, *it*, and nouns naming but one, be the subject of? *We*, *you*, *they*, and nouns naming more than one, be the subject of? Not classing *is* and *was* as *s*-forms, since the *s* in each is part of the root, which of the pronouns may be subjects of the *s*-form of verbs? Which class of nouns — those naming one, or those naming more than one — may be subjects of the *s*-form? What is said of *thou*? What then may you say of *art* and *hast*, above, which agree with *thou*? Of the verb-form of which *you* is the subject? What two very common errors in the use of verb-forms? How only can we guard against such errors and secure agreement of the verb with its subject?

Digitized by Google

Composition.

37

What is the foundation on which every sentence is built? May the subject be modified? What is a modifier? What is the modified subject? Illustrate both.

LESSON 21.

COMPOSITION.

We have here prepared the foundations of sentences which you are to complete by prefixing two or more suitable modifiers to each subject. Choose and arrange your modifiers so as to make neat, truthful, and sensible assertions.

Model. ————— eminence was reached.

That lofty eminence was reached.

- | | |
|----------------------------------|--------------------------|
| 1. — speaker was applauded. | 6. — houses are built. |
| 2. — difficulties were overcome. | 7. — soldiers perished. |
| 3. — leaf trembles. | 8. — opinions prevailed. |
| 4. — accident happened. | 9. — leader fell. |
| 5. — books should be read. | 10. — task is completed. |

For other subjects and predicates, the teacher is referred to Lessons 7 and 11.

Build sentences by prefixing modified subjects to the following predicates : —

- | | | |
|-------------------|------------------|------------------|
| 1. — frolic. | 4. — was caught. | 7. — flourished. |
| 2. — crawl. | 5. — escaped. | 8. — whistles. |
| 3. — are dashing. | 6. — chatter. | |

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

38

Graded Lessons in English.

Build, on each of the following subjects, three sentences similar to those in the model : —

Model. _____ sun _____.

The bright sun is shining.

The glorious sun has risen.

The unclouded sun is sinking.

1. — snow —. 2. — dew —. 3. — wind —.
4. — landscape —.

To the Teacher.—Please notice that the next Lesson begins with "Hints for Oral Instruction."

LESSON 22.

ADJECTIVES.

Hints for Oral Instruction.—You are now prepared to consider the fourth part of speech. The words that are added to the subject to modify its meaning are called **Adjectives**. In succeeding Lessons you will see that adjectives may be joined to nouns that are used otherwise than as subjects of sentences.

Some grammarians have formed a separate class of the little words *the*, and *an* or *a*, calling them **Articles**; but they may be classed as adjectives, for they are joined to nouns to modify their meaning.

I will write the word *boys* on the board, and you may name adjectives that will appropriately modify it. As

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Adjectives.

39

you give them, I will write these adjectives in a column, thus:—

Adjectives.

small
large
white
black
straight
crooked
five
some
all

boys.

What words here modify *boys* by adding the idea of size? What by adding the idea of color? What by adding the idea of form? What by adding the idea of number? What are such words called? Why?

Let the teacher name familiar objects and require the pupils to join appropriate adjectives to the names till their stock is exhausted.

DEFINITION.—An **Adjective** is a word used to modify a noun or a pronoun.

ANALYSIS AND PARSING.

Model.—*A fearful storm was raging.* Diagram and analyze as in Lesson 20.

Written Parsing.

Nouns.	Pronouns.	Adjectives.	Verbs.
storm	—	A fearful	was raging

Oral Parsing.—*A* is an adjective, because it is joined to the noun *storm* to modify its meaning; *fearful* is an adjective, because —; *storm* is a noun, because —; *was raging* is a verb, because —.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

40

Graded Lessons In English.

Analyze and diagram these sentences :—

1. The rosy morn advances.
2. The humble boon was obtained.
3. An unyielding firmness was displayed.
4. The whole earth smiles.
5. Several subsequent voyages were made.
6. That burly mastiff must be secured.
7. The slender greyhound was released.
8. The cold November rain is falling.
9. That valuable English watch has been sold.
10. I alone have escaped.
11. Both positions can be defended.
12. All such discussions should have been avoided.
13. That dilapidated old wooden building has fallen.

What adjectives in these sentences modify by expressing quality? What ones modify by pointing out? What ones, by numbering? _____

LESSON 23.

COMPOSITION.

Prefix five adjectives to each of the following nouns :—

Shrubs, wilderness, beggar, cattle, cloud.

Write ten sentences with modified subjects, using in each two or more of the following adjectives :—

A, an, the, heroic, one, all, many, every, either, first, tenth, frugal, great, good, wise, honest, immense, square, circular, oblong, oval, mild, virtuous, universal, sweet, careless, fragrant.

Digitized by Google

Composition.

41

Write five sentences with modified subjects, each of which shall contain one of the following words as subject:—

Chimney, hay, coach, robber, horizon.

Our knowledge of things is principally a knowledge of their qualities. A writer's style is largely affected by his choice and use of adjectives denoting these. We group a few denoting qualities perceived by the several senses. Join appropriate nouns to these:—

Seeing.

scarlet	dingy	gaudy
crimson	vivid	transparent

Hearing.

audible	deafening	discordant
loud	husky	melodious

Smelling.

fragrant	odorous	aromatic
----------	---------	----------

Tasting.

acid	delicious	palatable
pungent	insipid	luscious

Feeling.

rough	hard	tepid
dry	cold	hot

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

42

Graded Lessons in English.

It would be easy to add to these, especially to the first and the last class. Do so. Give some adjectives that denote intellectual qualities; some that denote moral qualities—pertaining to right and wrong.

An and *a* are forms of a word once spelled *an* and meaning one. After losing something of this force, *an* was still used before vowels and consonants alike; as, *an eagle, an ball, an hair, an use*. For the sake of ease in speaking, the word came later to have the two forms given above; *an* was retained before letters having vowel sounds, but dropped *n* and became *a* before letters having consonant sounds. This is the present usage.

Correct these errors :—

A apple; a obedient child; an brickbat; an busy boy.

Correct these errors :—

A heir; a hour; a honor.

Notice that the first letter of these words is silent.

Correct these errors :—

An unit; an utensil; an university; an ewe; an ewer; an union; an use; an history; an one-horse sled.

Unit begins with the sound of the consonant *y*; and *one*, with that of *w*.

Digitized by Google

Mention qualities belonging to each thing here named:—

chalk	ice	brooks	clouds
water	snow	ocean	music

Mention animals properly described by these adjectives:—

horned	fleet	cunning	ferocious
gentle	graceful	treacherous	venomous
faithful	useful	sagacious	ruminant

Careless persons and those with a meager list of adjectives at command overwork and abuse such words as *nice*, *awful*, *horrid*, *splendid*, *elegant*, *lovely*, and say *nice mountains*, *awful pens*, *horrid ink*, *splendid pie*, *elegant beef*, *lovely cheese*, etc.

Study the meaning of the last six adjectives, and use them to fill the following blanks:—

——— {	distinction	——— {	palace
	workmanship		victory
	calculation		illumination
——— {	stillness	——— {	manners
	chasm		taste
	rumbling		furniture
——— {	child	——— {	deeds
	features		dreams
	character		butchery

This work may very profitably be extended. It begets close observation of things and care and skill in describing them.

A word picture is often spoiled by using too many adjectives, as:—

*A great, large, roomy, spacious hall ;
Superb, delicious, magnificent pumpkin-pie ;
A stingy, miserly, close-fisted fellow.*

Omit those in italics, and notice how the description is improved. Subject some of your compositions to a like treatment, and note the gain.

LESSON 24.

MODIFIED PREDICATE.

Hints for Oral Instruction.—I will now show you how the predicate of a sentence may be modified.

“The ship *sails gracefully*.” What word is here joined to *sails* to tell the manner of sailing? P.—*Gracefully*.

T.—“The ship *sails immediately*.” What word is here joined to *sails* to tell the time of sailing? P.—*Immediately*.

T.—“The ship *sails homeward*.” What word is here joined to *sails* to tell the direction of sailing? P.—*Homeward*.

T.—These words *gracefully*, *immediately*, and *homeward* are modifiers of the predicate. In the first sentence, *sails gracefully* is the **Modified Predicate**.

Let the following modifiers be written on the board as the pupil suggests them:—

Modified Predicate.

45

The ship sails	{	instantly.	Which words indicate the time of sailing? Which, the place or direction? Which, the manner?
		soon.	
		daily.	
		hither.	
		hence.	
		there.	
		rapidly.	
		smoothly.	
		well.	

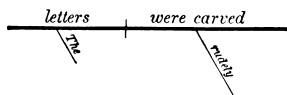
The teacher may suggest predicates, and require the pupils to find as many appropriate modifiers as they can.

The Predicate with its modifiers is called the **Modified Predicate**.

ANALYSIS AND PARSING.

Analyze and diagram the following sentences, and parse the nouns, pronouns, verbs, and adjectives : —

Model. — *The letters were rudely carved.*



Written Parsing. — See *Model*, Lesson 22.

Oral Analysis. — This is a sentence, because — ; *letters* is the subject, because — ; *were carved* is the predicate, because — ; *The* is a modifier of the subject, because — ; *rudely* is a modifier of the predicate, because — ; *The letters* is the modified subject, *were rudely carved* is the modified predicate.

1. He spoke eloquently.
2. She chattered incessantly.
3. They searched everywhere.
4. I shall know presently.
5. The bobolink sings joyously.
6. The crowd cheered heartily.
7. A great victory was finally won.
8. Threatening clouds are moving slowly.
9. The deafening waves dash angrily.
10. These questions may be settled peaceably.
11. The wounded soldier fought bravely.
12. The ranks were quickly broken.
13. The south wind blows softly.
14. Times will surely change.
15. An hour stole on.

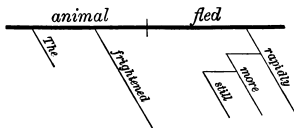
LESSON 25.

ANALYSIS AND PARSING.

ONE MODIFIER JOINED TO ANOTHER.

Analyze and diagram the following sentences, and parse the nouns, pronouns, adjectives, and verbs : —

Model. — *The frightened animal fled still more rapidly.*



The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Review Questions.

47

Explanation of the Diagram. — Notice that the three lines forming this group all slant the same way to show that each stands for a modifying word. The line standing for the principal word of the group is joined to the predicate line. The end of each of the other two lines is broken, and turned to touch its principal at an angle.

Oral Analysis. — This is a sentence, because — ; *animal* is the subject, because — ; *fled* is the predicate, because — ; *The* and *frightened* are modifiers of the subject, because — ; *still more rapidly* is a modifier of the predicate, because it is a group of words joined to it to limit its meaning ; *rapidly* is the principal word of the group ; *more* modifies *rapidly*, and *still* modifies *more* ; *The frightened animal* is the modified subject ; *fled still more rapidly* is the modified predicate.

1. The crocus flowers very early.
2. A violet bed is budding near.
3. The Quakers were most shamefully persecuted.
4. Perhaps he will return.
5. We laughed very heartily.
6. The yellow poplar leaves floated down.
7. The wind sighs so mournfully.
8. Few men have ever fought so stubbornly.
9. The debt will probably be paid.
10. The visitor will soon be here.
11. That humane project was quite generously sustained.
12. A perfectly innocent man was very cruelly persecuted.

REVIEW QUESTIONS.

What is an adjective? What are the words *an* or *a*, and the called by some grammarians? What may they be called? When is *a* used, and when *an*? Give examples of their misuse. Correct them, and give your reasons. Adjectives modify by expressing what?

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

48

Graded Lessons in English.

What grows out of the careless use, or of a scanty list, of adjectives? What is said of some overworked adjectives? Of a superabundance of adjectives?

What is the modified predicate? Give an example. Give an example of one modifier joined to another.

LESSON 26.

Select your subjects from Lesson 9, and construct twenty sentences having modified subjects and modified predicates.

IMPROMPTU EXERCISE.

Select sentences from Lessons 6, 7, and 11, and conduct the exercise as directed in Lesson 10. Let the struggle be to see who can supply the greatest number of modifiers of the subject and of the predicate. The teacher can vary this exercise.

LESSON 27.

ADVERBS.

Hints for Oral Instruction. — You have learned, in the preceding Lessons, that the meaning of the predicate may be qualified by modifiers, and that one modifier may be joined to another. Words used to modify the predicate of a sentence and those used to modify modifiers belong to one class, or one part of speech, and are called **Adverbs**.

T. — "She decided *too hastily*." What word tells how she decided? **P.** — *Hastily*. **T.** — What word tells how

Digitized by Google

hastily? P. — *Too*. T. — What then are the words *too* and *hastily*? P. — Adverbs.

T. — "*Too much* time has been wasted." What word modifies *much* by telling how much? P. — *Too*.

T. — What part of speech is *much*? P. — An adjective.

T. — What then is *too*? P. — An adverb.

T. — Why is *too* in the first sentence an adverb? Why is *too* in the second sentence an adverb? Why is *hastily* an adverb?

Let the teacher use the following and similar examples, and continue the questions: "He thinks *so*;" "*So much* time has been wasted."

Let the teacher give verbs, adjectives, and adverbs, and require the pupils to modify them by appropriate adverbs.

DEFINITION. — An **Adverb** is a word used to modify a verb, an adjective, or an adverb.

ANALYSIS AND PARSING.

Analyze, diagram, and parse the following sentences.

Model. — *We have been very agreeably disappointed.* Diagram as in Lesson 25.

For **Written Parsing**, use *Model*, Lesson 22, adding a column for adverbs.

Oral Parsing. — *We* is a pronoun, because —; *have been disappointed* is a verb, because —; *very* is an adverb, because it is joined to the adverb *agreeably* to tell how agreeably; *agreeably* is an adverb, because it is joined to the verb *have been disappointed* to indicate manner.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

50

Graded Lessons in English.

1. The plow-boy plods homeward.
2. The water gushed forth.
3. Too much time was wasted.
4. She decided too hastily.
5. You should listen more attentively.
6. More difficult sentences must be built.
7. An intensely painful operation was performed.
8. The patient suffered intensely.
9. That story was peculiarly told.
10. A peculiarly interesting story was told.
11. An extravagantly high price was paid.
12. That lady dresses extravagantly.

What adverbs in these sentences modify by expressing (1) manner, (2) degree, and (3) place or direction ?

The pupil will notice that, in some of the examples above, the same adverb modifies an adjective in one sentence and an adverb in another; and that, in other examples, an adjective and a verb are modified by the same word. You learn from this why such modifiers are grouped into one class.

LESSON 28.

ANALYSIS AND PARSING.

MISCELLANEOUS EXAMPLES FOR REVIEW.

1. You must diagram neatly.
2. The sheaves are nearly gathered.
3. The wheat is duly garnered.
4. The fairies were called together.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Composition.

51

5. The birds chirp merrily.
6. This reckless adventurer has returned.
7. The wild woods rang.
8. White, fleecy clouds are floating above.
9. Those severe laws have been repealed.
10. A republican government was established.
11. An unusually large crop had just been harvested.
12. She had been waiting quite patiently.
13. A season so extremely warm had never before been known.
14. So brave a deed¹ cannot be too warmly commended.

LESSON 29.

COMPOSITION.

MISCELLANEOUS EXERCISES FOR REVIEW.

Build sentences containing the following adverbs :—

Hurriedly, solemnly, lightly, well, how, somewhere, abroad, forever, seldom, exceedingly.

Using the following subjects and predicates as foundations, build six sentences having modified subjects and modified predicates, two of which shall contain adverbs modifying adjectives; two, adverbs modifying adverbs; and two, adverbs modifying verbs.

- | | |
|-----------------------------|-------------------------------|
| 1. — boat glides —. | 4. — elephant was captured —. |
| 2. — cloud is rising —. | 5. — streams flow —. |
| 3. — breezes are blowing —. | 6. — spring has opened —. |

¹ *Can be commended* is the verb, and *not* is an adverb.

We here give you, in classes, the material out of which you are to build five sentences with modified subjects and modified predicates.

Select the subject and the predicate first:—

<i>Nouns and Pronouns.</i>	<i>Verbs.</i>	<i>Adjectives.</i>	<i>Adverbs.</i>
branch	was running	large, that	lustily
coach	were played	both, the	downward
they	cried	all, an	very
we	is growing	several, a	rapidly
games	cheered	amusing	not, loudly, then

LESSON 30.

ERRORS FOR CORRECTION.

Caution.—When two or more adjectives are used with a noun, care must be taken in their arrangement. If there is any difference in their relative importance, place nearest the noun the one that is most intimately connected with it.

To the Teacher.—We have in mind here those numerous cases where one adjective modifies the noun, and the second modifies the noun as limited by the first. "*All ripe apples are picked.*" Here *ripe* modifies *apples*, but *all* modifies *apples* limited by *ripe*. Not all apples are picked, but only all that are ripe.

Correct the following errors of position:—

A wooden pretty bowl stood on the table.
The blue beautiful sky is cloudless.
A young industrious man was hired.
The new marble large house was sold.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Caution.—When the adjectives are of the same rank, place them where they will sound the best. This will usually be in the order of their length—the longest last.

Correct these errors :—

An entertaining and fluent speaker followed ; An enthusiastic, noisy, large crowd was addressed.

Caution.—Do not use the pronoun *them* for the adjective *those*.

Correct these errors :—

Them books are nicely bound ; Them two sentences should be corrected.

Pupils may be required to copy choice selections from literature, and to note carefully capitals, punctuation, and the use of adjectives, etc. We offer the following exercise as a specimen :—

We piled with care our nightly stack
Of wood against the chimney-back,—
The oaken log, green, huge, and thick,
And on its top the stout back-stick ;
The knotty fore-stick laid apart,
And filled between with curious art
The ragged brush ; then, hovering near,
We watched the first red blaze appear,
Heard the sharp crackle, caught the gleam
On whitewashed wall and sagging beam,
Until the old, rude-furnished room
Burst, flower-like, into rosy bloom.

Whittier. — Snow-Bound.



Of what are the lines, above, a picture? Where and in what kind of house, do you think this picture was seen?

What object is pictured by the help of five adjectives? Are the adjectives that precede the name of this object of the same rank? Are those that follow of the same rank? What noun is modified by three adjectives of different rank? What noun by three adjectives two of which are of the same rank? What difference is found in the punctuation of these several groups? Are there any groups of words used to modify verbs or nouns? If there are and you can find them, show what words they modify.

Notice how the noun *crackle* crackles as you pronounce

Review Questions.

55

it, and how the adjective *sharp* makes it penetrate. Notice how strong a picture is made in the two lines immediately before the last.

Why does Whittier use *nightly* in line 1? What does *stout* in line 4 mean? What is understood after *between* in line 6? What propriety in calling brush *ragged* in line 7? What does *sagging* in line 10 suggest? What color does *rosy* in the last line denote? Are all roses of one color?

The adjectives here used bring out the most prominent qualities of the room, and these qualities bring along with them into the imagination all the other qualities. This is what we must try to make our adjectives do.

Point out the adjectives in the selection above, and explain the office of each.

What peculiar use of capitals do you discover in these lines of poetry?

Much that has been suggested above concerning the use of adjectives will apply to adverbs also.

REVIEW QUESTIONS.

What is an adverb? Give an example of an adverb modifying an adjective; one modifying a verb; one modifying an adverb. Why are such expressions as *a wooden pretty bowl* faulty? Why is an *enthusiastic, noisy, large crowd* faulty? Why is *them books* wrong?

Thus far we have been dealing with sentences not connected with other sentences. But we seldom find them standing thus apart and alone. They are usually grouped in **paragraphs**—each sentence of the group helping to

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

56

Graded Lessons in English.

develop, and all together developing, the general thought of the paragraph. This their joint work relates them one to another, and gives them properties which they would not have if they stood alone.

To understand sentences fully then we must study them in paragraphs; to master their construction, we must compose them in paragraphs. To be known and handled as parts of a whole, the whole must be studied.

COMPOSITION OF SENTENCES IN PARAGRAPHS.

SELECTION FROM DARWIN.

Morren says that angleworms often lie for hours almost motionless close beneath the mouths of their burrows. I have occasionally noticed the same fact with worms kept in pots in the house; so that by looking down into their burrows their heads could just be seen. If the ejected earth or rubbish over the burrows be suddenly removed, the end of the worm's body may very often be seen rapidly retreating.

This habit of lying near the surface leads to their destruction to an immense extent. Every morning, during certain seasons of the year, the thrushes and blackbirds on all the lawns throughout the country draw out of their holes an astonishing number of worms; and this they could not do unless they lay close to the surface.

It is not probable that worms behave in this manner for the sake of breathing fresh air, for they can live for a long time under water. I believe that they lie near the surface for the sake of warmth, especially in the morning; and we shall hereafter find that they often coat the mouths of their burrows with leaves, apparently to prevent their bodies from coming into close contact with the cold, damp earth.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

The Uses of Words and Groups of Words. — We will break up Mr. Darwin's first group of sentences into single sentences or single statements, each having but one predicate verb.

1. Angeworms often lie for hours almost motionless close beneath the mouths of their burrows. 2. Morren says this. 3. I have occasionally noticed the same fact with worms kept in pots in the house. 4. By looking down into their burrows their heads could just be seen. 5. The ejected earth or rubbish over the burrows may suddenly be removed. 6. The end of the worm's body may then very often be seen rapidly retreating.

Find the two chief words (subject and predicate) in 1. What does *often* do? What does the group of words *for hours* do? The group *almost motionless* describes what things? The group *close beneath the mouths of their burrows*, used like a single adverb, tells what? Find the two chief words in 2. *This* helps out the meaning of *says*, but it is not an adverb. *This* is a pronoun standing here for the thing said. What whole sentence does *this* take the place of? Find the subject and the predicate verb in 3. What noun follows this verb to tell what Mr. Darwin noticed? What does *occasionally* do? What does *same* go with? What group of eight words tells in what way Mr. Darwin noticed this fact? Find the unmodified subject and predicate in 4. What does the second *their* go with? What does *by looking down into their burrows* tell? What does *just* do? In 5, put *what* before *may be removed*, and find two words either of which may be used as subject. What is the office of *the, ejected*, and the group *over the burrows*? What does *suddenly* do? Find the subject and the predicate verb in 6. *Retreating* helps out the meaning of the predicate and at the same time modifies the subject. Notice that *the end rapidly retreating* is not a sentence, nor is *worms kept in pots*, in 3. *Retreating* and *kept* here express action, but they are not predicates; they do not assert.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

You learned in Lesson 16 that certain forms of the verb do not assert. *Of the worm's body* modifies *what*? *Then* and *very often* do what?

If you will compare these numbered sentences with Mr. Darwin's, you will see how two or more sentences are put together to make one longer sentence. You see Mr. Darwin puts our sentence 1 after *says* to tell what Morren says. What word here helps to bring two sentences together? Change this sentence about so as make *says Morren* come last. See how many other changes you can make in the arrangement of the words and groups of words in this sentence. What two words are used to join 3 and 4 together? Notice that these sentences are not joined so closely as 1 and 2, as is shown by the semicolon. Notice that *if* has much to do in joining 5 and 6. These are more closely joined than 3 and 4, but not so closely as 1 and 2. How is this shown by the punctuation? Put 5 and 6 together and change their order. Find, if you can, still another arrangement.

To the Teacher.—It is very important that pupils should learn to see words in groups and to note their offices. If difficulties and technicalities be avoided, such exercises as we suggest above may be begun very early. They will lead to an intelligent observation of language and will prepare the way for the more formal lessons of the text-book.

If time can be had, such exercises may profitably be continued through the second and third paragraphs of the selection above.

The Paragraph.—If we write about only one thing or one point, our sentences will be closely related to each other. If we write on two or more points, there will be two or more sets of sentences—the sentences of each set closely related one to another, but the sets themselves not so closely related. A group of sentences expressing what we have to say on a single point, or division, of our subject is called a **paragraph**. How many paragraphs do you find in the selection above? How are they separated on the page?

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Let us examine this selection more carefully to find whether the sentences of each group are all on a single point and closely related, and whether the groups themselves are related. Do the sentences of the first paragraph all help to tell of a certain habit of angleworms? Do the sentences of the second paragraph tell what results from this habit? Do the sentences of the third paragraph tell what is thought to be the cause of this habit? If you can say *yes* to these questions, the sentences in each paragraph must be closely related. Are a habit, a result of it, and a cause of it related in thought, or meaning? If so, the paragraphs are related. In the fewest words needed, tell what this habit is, what the result of it is, and what the cause of it is.

You must now see that paragraphing helps the writer in planning his production and arranging his matter, and helps the reader to understand what the writer has done.

The Style.—We shall not here say much about what we may call the style of the author—his way of putting his thought, or manner of expressing it. But this you will notice: his words are few, plain, and simple; the arrangement of them is easy; and hence what is said is said clearly. You are nowhere in doubt about his meaning unless it be in the second paragraph. It may puzzle you to see what *their*, *they*, and *they* in the second sentence of this paragraph stand for. Transpose *an astonishing number of worms* and *out of their holes*, and substitute *birds* and *worms* for *they* and *they*, and see whether the meaning would not be clearer. Clearness is worth all it costs. You cannot take too much pains to be understood.

First-hand Knowledge.—As you know, we get our knowledge in two ways. We get it by seeing, and thinking about what we see; and we get it by listening to other people and reading what they have written. What we get by seeing, by observation, is first-hand knowledge; what we get from others is second-hand knowledge.

Both kinds are useful; we cannot have too much of either. But the kind that it does us most good to get and is worth most to us when got is first-hand knowledge. This especially is the kind which you should make your compositions of. In the first two paragraphs of the selection above, Darwin is telling what he saw, and in the third he is explaining what he saw. That is why what he says is so fresh and interesting.

And just one thing more. If such a man as Charles Darwin thought it worth his while to spend much time in studying and experimenting upon angleworms and then to write a large book about them, surely you need not think anything in nature beneath your notice.

ORIGINAL COMPOSITION.

In two or three short paragraphs, tell what you have observed of some worm, insect, or other creature, and what you think about it.

To the Teacher. — We suggest that what is said above be read by the pupils and discussed in the class, and that the substance of it be reproduced in the pupils' own language. Such reproduction will serve as a lesson in oral composition.

It may be profitable for the pupils to reproduce the selection from Darwin.

LESSON 31.

PHRASES INTRODUCED BY PREPOSITIONS.

Hints for Oral Instruction. — In Lessons 25 and 27, you learned that several words may be grouped together and used as one modifier. In the examples there given, the principal word is joined directly to the subject or to the predicate, and this word is modified by another word. In Lesson 30 and in this, groups of words are used as

modifiers, but these words are not united with one another, or with the word which the group modifies, as they are in the preceding Lessons.

I will write on the board this sentence: "De Soto marched *into Florida*."

T.—What tells where De Soto marched? P.—*Into Florida*. T.—What is the principal word of the group? P.—*Florida*. T.—Is *Florida* joined directly to the predicate, as *rapidly* was in Lesson 25? P.—No. T.—What little word comes in to unite the modifier to *marched*? P.—*Into*. T.—Does *Florida* alone, tell where he marched? P.—No. T.—Does *into* alone, tell where he marched? P.—No.

T.—These groups of related words are called **Phrases**. Let the teacher draw on the board the diagram of the sentence above.

Phrases of the form illustrated in this diagram are the most common, and they perform a very important function in our language.

Let the teacher frequently call attention to the fact that all the words of a phrase are taken together to perform one distinct office.

A phrase modifying the subject is equivalent to an adjective, and frequently may be changed into one. "The dew *of the morning* has passed away." What word may be used for the phrase, *of the morning*? P.—*Morning*. T.—Yes. "The *morning* dew has passed away."

A phrase modifying the predicate is equivalent to an adverb, and frequently may be changed into one. "We shall go *to that place*." What word may be used for the phrase, *to that place*? P.—*There*. T.—Yes. "We shall go *there*."

Change the phrases in these sentences:—

A citizen *of America* was insulted; We walked *toward home*.

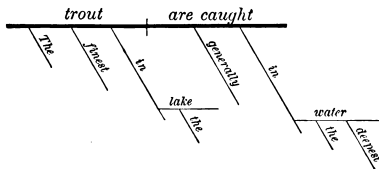
Let the teacher write on the board the following words, and require the pupils to add to each, one or more words to complete a phrase, and then to construct a sentence in which the phrase is properly employed: *to, from, by, at, on, with, in, into, over*.

DEFINITION.—A **Phrase** is a group of words denoting related ideas but not expressing a thought.

ANALYSIS AND PARSING.

Analyze the following sentences, and parse the nouns, pronouns, adjectives, verbs, and adverbs.

Model.—*The finest trout in the lake are generally caught in the deepest water.*



The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Explanation of the Diagram.— You will notice that the diagram of the phrase is made up of a slanting line standing for the introductory and connecting word, and a horizontal line representing the principal word. Under the latter are placed the little slanting lines standing for the modifiers of the principal word. Here and elsewhere all modifiers are joined to the principal words by slanting lines.

Oral Analysis.— This is a sentence, because — ; *trout* is the subject, because — ; *are caught* is the predicate, because — ; the words *The* and *finest*, and the phrase *in the lake* are modifiers of the subject, because — ; the word *generally* and the phrase *in the deepest water* are modifiers of the predicate, because — ; *in* introduces the first phrase, and *lake* is the principal word ; *in* introduces the second phrase, and *water* is the principal word ; *the* and *deepest* are modifiers of *water* ; *The finest trout in the lake* is the modified subject, and *are generally caught in the deepest water* is the modified predicate.

1. The gorilla lives in Africa.
2. It seldom rains in Egypt.
3. The Pilgrims landed at Plymouth.
4. The wet grass sparkled in the light.
5. The little brook ran swiftly under the bridge.
6. Burgoyne surrendered at Saratoga.
7. The steeples of the village pierced through the dense fog.
8. The gloom of winter settled down on everything.
9. A gentle breeze blows from the south.
10. The temple of Solomon was destroyed.
11. The top of the mountain is covered with snow.
12. The second Continental Congress convened at Philadelphia.

Name the phrases of place in these sentences ; the verbs modified by adverbs and by phrases ; the nouns modified by phrases ; the sentences containing each two phrases.

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"

64

Graded Lessons in English.

LESSON 32.

COMPOSITION.

Build sentences, employing the following phrases as modifiers:—

To Europe, of oak, from Albany, at the station, through the fields, for vacation, among the Indians, of the United States.

Prefix to the following predicates subjects modified by phrases:—

— is situated on the Thames.	— was received.
— has arrived.	— has just been completed.
— was destroyed by an earthquake.	— may be enjoyed.

Add to the following subjects predicates modified by phrases:—

Iron —.	The Bible —.	Paul —.
The trees —.	Sugar —.	Strawberries —.
Squirrels —.	Cheese —.	The mountain —.

Write five sentences, each of which shall contain one or more phrases used as modifiers.

You have all been on picnics and know a great deal about them. You know what they are and what they are for; to what places they are excursions, who go, how they go, what is carried along, what games are played, how the feast is served and eaten, what fun and recreation and enjoyment are had, and how tired everybody gets!

Study the picture minutely; name the three features of a picnic which you think are most enjoyable, and expand

Digitized by Google

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"

Composition.

65



"THE CHILDREN'S PICNIC."

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

66

Graded Lessons In English.

these three headings, or sub-topics, into three paragraphs, which when put together make a composition.

To the Teacher.—See that the paragraphs of the composition fairly exhaust the thought of the head, or sub-topic, that they stand in proper order, and that they are composed of sentences varied in kind and length.

Allow for the individuality of your pupils in the selection and in the grouping of the matter.

LESSON 33.

COMPOSITION.

Rewrite the following sentences, changing the italicized words into equivalent phrases : —

Model.—A *golden* image was made = An image *of gold* was made.

You notice that the adjective *golden* is placed before the subject, but, when changed to a phrase, it follows the subject.

1. The book was *carefully* read.
2. The old soldiers fought *courageously*.
3. A group of children were strolling *homeward*.
4. No season of life should be spent *idly*.
5. The *English* ambassador has just arrived.
6. That *generous* act was liberally rewarded.

Rewrite the following sentences, changing phrases to adjectives or adverbs, and most of the adjectives and adverbs to phrases : —

1. A thing of beauty is a joy forever.
2. English grammar is remarkably simple.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Prepositions.

67

3. In all cases vulgarisms are to be shunned.
4. The conclusions of science are sometimes only highly probable.
5. The word *demijohn* has sadly puzzled people.

Change the following adjectives and adverbs into equivalent phrases, and employ the phrases in sentences of your own : —

Wooden, penniless, eastward, somewhere, here, evening, everywhere, yonder, joyfully, wintry.

Make a sentence out of the words in each line below : —

Boat, waves, glides, the, the, over.

He, Sunday, church, goes, the, on, to.

Year, night, is dying, the, the, in.

Qualities, Charlemagne, vices, were alloyed, the, great, of, with.

Indians, America, intemperance, are thinned, the, out, of, by.

LESSON 34.

PREPOSITIONS.

Hints for Oral Instruction. — In the preceding Lessons, the little words placed before nouns and with them forming phrases belong to a class of words called **Prepositions**. You noticed that these words, which you have now learned to call prepositions, introduce phrases. The preposition shows the relation of the thing denoted by the principal word of the phrase to that of the word which the phrase modifies. It serves also to connect these words.

In the sentence, "The squirrel *ran up a tree*," what

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

68

Graded Lessons in English.

word shows the relation of the act of running, to the tree?
Ans. *Up*.

Other words may be used to express different relations. Repeat, nine times, the sentence above given, supplying in the place of *up* each of the following prepositions: *around, behind, down, into, over, through, to, under, from*.

Let this exercise be continued, using such sentences as, "The man went *into the house*;" "The ship sailed *toward the bay*."

DEFINITION. — A **Preposition** is a word that introduces a phrase modifier, and shows the relation, in sense, of its principal word to the word modified.

ANALYSIS AND PARSING.

Model. — *Flowers preach to us*.

For **analysis** and **diagram**, see Lesson 31.

For **written parsing**, see Lesson 22. Add the needed columns.

Oral Parsing. — *Flowers* is a noun, because — ; *preach* is a verb, because — ; *to* is a preposition, because it shows the relation, in sense, between *us* and *preach*; *us* is a pronoun, because it is used instead of the name of the speaker and the names of those for whom he speaks.

1. The golden lines of sunset glow.
2. A smiling landscape lay before us.
3. Columbus was born at Genoa.
4. The forces of Hannibal were routed by Scipio.
5. The capital of New York is on the Hudson.
6. The ships sail over the boisterous sea.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Impromptu Exercise.

69

7. All names of the Deity should begin with capital letters.
8. Air is composed chiefly of two invisible gases.
9. The greater portion of South America lies between the tropics.
10. The laurels of the warrior must at all times be dyed in blood.
11. The first word of every entire sentence should begin with a capital letter.
12. The subject of a sentence is generally placed before the predicate.

The words and the phrases in the sentences above are in what we call their **natural order**. From any of these sentences determine the natural order (1) of subject and predicate, and (2) of the phrase and the word it modifies; from 1, 6, 7, 8, and 11, determine the natural order of (3) adjectives and the nouns they modify; and from 8, 10, 11, and 12, determine the places an adverb or a phrase may hold with respect to its verb (4) when this is made up of two or more words.

If placed out of their natural order, words and phrases are said to be **transposed**; and, as we shall see, this may involve the use of the comma.

IMPROMPTU EXERCISE.

Let the teacher write on the board a subject and a predicate that will admit of many modifiers. The pupils are to expand the sentence into as many sentences as possible, each containing one apt phrase modifier. The competition is to see who can build the most and the best sentences in a given time. The teacher gathers up the slates and reads the work aloud, or has the pupils exchange slates and read it themselves.

Digitized by Google

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"

70

Graded Lessons In English.

LESSON 35.

COMPOUND SUBJECT AND COMPOUND PREDICATE.

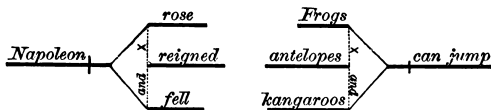
When two or more subjects united by a connecting word have the same predicate, they form a **Compound Subject**; and, when two or more predicates connected in like manner have the same subject, they form a **Compound Predicate**.

In the sentence, "*Birds and bees* can fly," the two words *birds* and *bees*, connected by *and*, have the same predicate; the same action is asserted of both birds and bees. In the sentence, "*Leaves fade and fall*," two assertions are made of the same things. In the first sentence, *birds* and *bees* form the compound subject; and, in the second, *fade* and *fall* form the compound predicate.

Analyze the following sentences, and parse the words:—

Models. — *Napoleon rose, reigned, and fell.*

Frogs, antelopes, and kangaroos can jump.



Explanation of the Diagram. — The short line following the subject line represents the entire predicate, and is supposed to be continued in the three horizontal lines that follow, each of which represents one of the parts of the compound predicate. These three lines are united

Digitized by Google

Review Questions.

71

by dotted lines, which stand for the connecting words. The \times denotes that an *and* is understood.

Study this explanation carefully, and you will understand the other diagram.

Oral Analysis of the first sentence.

This is a sentence, because — ; *Napoleon* is the subject, because — ; *rose*, *reigned*, and *fell* form the compound predicate, because they belong in common to the same subject, and say something about Napoleon. *And* connects *reigned* and *fell*.

1. The Rhine and the Rhone rise in Switzerland.
2. Time and tide wait for no man.
3. Washington and Lafayette fought for American independence.
4. Wild birds shrieked, and fluttered on the ground.
5. The mob raged and roared.
6. The seasons came and went.
7. Pride, poverty, and fashion cannot live in the same house.
8. The tables of stone were cast to the ground and broken.
9. Silver or gold will be received in payment.
10. Days, months, years, and ages will circle away.

REVIEW QUESTIONS.

What is a phrase? A phrase modifying a subject is equivalent to what? Illustrate. A phrase modifying a predicate is equivalent to what? Illustrate.

What are prepositions? Give the definition. What is the natural order of subject and predicate? Of a phrase and the word it modifies? Of adjectives and their nouns? Of an adverb and the verb it modifies when this is one word? When two or more words? What do you understand by a compound subject? Illustrate. What do you understand by a compound predicate? Illustrate.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

72

Graded Lessons In English.

LESSON 36.

CONJUNCTIONS AND INTERJECTIONS.

The words *and* and *or*, used in the preceding Lesson to connect the nouns and the verbs, belong to a class of words called **Conjunctions**.

Conjunctions may connect words used as modifiers also, as :—

A daring *but* foolish feat was performed.

They may connect phrases, as:—

We shall go to Saratoga *and* to Niagara.

They may connect clauses—that is, expressions that, standing alone, would be sentences, as:—

He must increase, *but* I must decrease.

DEFINITION.—A **Conjunction** is a word used to connect words, phrases, or clauses.

The **Interjection** is the eighth and last part of speech. Interjections are mere exclamations, and are without grammatical relation to any word in the sentence.

DEFINITION.—An **Interjection** is a word used to express strong or sudden feeling.

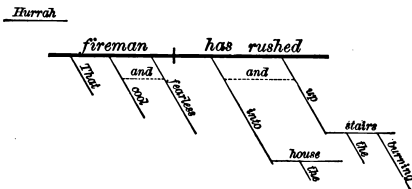
Examples:—

Bravo! hurrah! pish! hush! ha, ha! alas! hail! lo! pshaw!

Digitized by Google

Analyze the following sentences, and parse the words:—

Model.— *Hurrah! that cool and fearless fireman has rushed into the house and up the burning stairs.*



Explanation of the Diagram.—The line representing the interjection is not connected with the diagram. Notice the dotted lines, one standing for the *and* which connects the two word modifiers; the other, for the *and* connecting the two phrase modifiers.

Written Parsing.

N.	Pro.	Adj.	Vb.	Adv.	Prep.	Conj.	Int.
fireman		the	has rushed		into	and	hurrah
house		that			up	and	
stairs		cool					
		fearless					
		burning					

Oral parsing of the conjunction and the interjection.

The two *ands* are conjunctions, because they connect. The first connects two word modifiers; the second, two phrase modifiers. *Hurrah* is an interjection, because it expresses a burst of sudden feeling.

1. The small but courageous band was finally overpowered.
2. Lightning and electricity were identified by Franklin.
3. A complete success or an entire failure was anticipated.
4. Good men and bad men are found in all communities.
5. Vapors rise from the ocean and fall upon the land.
6. The Revolutionary War began at Lexington and ended at

Yorktown.

7. Alas ! all hope has fled.
8. Ah ! I am surprised at the news.
9. Oh ! we shall certainly drown.
10. Pshaw ! you are dreaming.
11. Hurrah ! the field is won.

Were identified in 2 is asserted of two things ; *rise* and *fall* in 5, of two or more. *Was anticipated* in 3 is asserted of only one thing, — success or failure, — and *has fled* in 7, of only one.

Singular means one, **plural** means more than one, and **agreement** means that plural subjects have plural verbs, and subjects in the singular have verbs in the singular. Two or more subjects in the singular connected by *and* and naming different things make a plural subject; and two or more subjects in the singular connected by *or*, *nor*, *either . . . or*, *neither . . . nor* make a subject in the singular.

The adjectives *each*, *every*, and *no*, belonging to nouns in the singular, show that the things named are taken separately, and that the verb must be in the singular.

Remembering now that nouns with **s-ending** are plural

and that verbs with s-ending are singular, justify the italicized verb-forms in these sentences:—

1. Each word and gesture *was* suited to the thought.
2. In the death of Franklin, a philosopher and statesman *was* lost to the world.
3. Beauty and utility *are* combined in nature.
4. Either beauty or utility *appears* in every natural object.
5. Here *is* neither beauty nor utility.
6. Time and tide *wait* for no man.
7. Wisdom and prudence *dwell* with the lowly man.
8. *Does* either landlord or tenant profit by this bill?
9. Neither landlords nor tenants *profit* by this bill.
10. Every fly, bee, beetle, and butterfly *is* provided with six feet.
11. That desperate robber and murderer *was* finally secured.
12. Every bud, leaf, and blade of grass *rejoices* after the warm rain.
13. That desperate robber and that murderer *were* finally secured.
14. The builder and owner of the yacht *has* sailed from Liverpool.
15. The builder and the owner of the yacht *have* sailed from Liverpool.
16. A lame and blind man *was* provided with food and lodging.
17. A lame and a blind man *were* provided with food and lodging.
18. No dew, no rain, no cloud *comes* to the relief of the parched earth.

Select the sentences with subjects in the singular connected by *and*, naming (1) different things, and (2) the same thing; (3) connected by *or*, *nor*, *either . . . or*, *neither . . . nor*; and (4) modified by *each*, *every*, and *no*. Point out the effect of repeating *that*, *the*, or *a* in 13, 15, and 17.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

76

Graded Lessons In English.

1. Neither John nor his *sisters* were there.
2. *Action*, and not words, *is* needed.
3. *Bread and milk* *is* good food.
4. The *committee* are unable to agree on *their* report.
5. The *committee* *has* made *its* report.

Pupils will see, in examples like 1 above, that the verb agrees with its nearest subject, and that the plural subject is usually placed next the verb; in examples like 2, that the verb agrees with the affirmative subject, another verb being understood with the negative subject; that in 3, *bread and milk* represents one article of food; and that in 4, the individuals of the committee are thought of, while in 5, the committee as a whole is thought of. In 4 and 5, the agreement of the pronoun also may be noted. Pronouns may be introduced into many of the preceding exercises and the pupils led to apply to the agreement of the pronoun with its antecedent what has been learned of the agreement of the verb with its subject. Let the pupils determine why the following connected subjects are arranged in the proper order:—

You and I are invited.	You and Mary are invited.
Mary and I are invited.	You and Mary and I are invited.

LESSON 37.

PUNCTUATION AND CAPITAL LETTERS.

COMMA—RULE.—Phrases that are placed out of their natural order and made emphatic, or that are loosely connected with the rest of the sentence, should be set off by the comma.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Punctuate the following sentences :—

Model. — The cable, *after many failures*, was successfully laid.

Upon the platform 'twixt eleven and twelve I'll visit you.
To me this place is endeared by many associations.
Your answers with few exceptions have been correctly given.
In English much depends on the placing of phrases.

COMMA — RULE. — Words or phrases connected by conjunctions are separated from each other by the comma unless all the conjunctions are expressed.

Punctuate the following sentences :—

Model. — *Cæsar came, saw, and conquered.*
Cæsar came and saw and conquered.
He traveled in *England, in Scotland, and in Ireland.*

Tell why the comma is used in the first and third sentences but not in the second.

A brave prudent and honorable man was chosen.
Augustus Tiberius Nero and Vespasian were Roman emperors.
Through rainy weather across a wild country over muddy roads after a long ride we came to the end of our journey.

PERIOD AND CAPITAL LETTER — RULE. — Abbreviations generally begin with capital letters and are always followed by the period.

Correct the following errors, and (see list at the end of the book) tell what these abbreviations stand for:—

Model. — *Mr., Esq., N. Y., P. M.*

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

78

Graded Lessons in English.

gen, a m, mrs, no, u s a, n e, eng, p o, rev, prof, dr, gram, capt, col, co, va, conn, feb, n o, n, oct, pres, sat, vt, apr, ky, a d, gov, wed, s, w, treas, maj, sec, geo, hon.

Pick out from the list of abbreviations a score of the most common ones that do not begin with capital letters, and tell what they stand for.

EXCLAMATION POINT—RULE.—All **exclamatory expressions** must be followed by the exclamation point.

Punctuate the following expressions: —

Model. — *Ah! Oh! Zounds! Stop pinching!*

Pshaw, whew, alas, ho Tom, hallo Sir, good-by, welcome.

LESSON 38.

COMPOSITION.

Write predicates for the following compound subjects: —

Snow and hail ; leaves and branches ; a soldier or a sailor ; London and Paris.

Write compound predicates for the following subjects: —

The sun ; water ; fish ; steamboats ; soap ; farmers ; fences ; clothes.

Write subjects for the following compound predicates: —

Live, feel, and grow ; judges and rewards ; owes and pays ; inhale and exhale ; expand and contract ; flutters and alights ; fly, buzz, and sting ; restrain or punish.

Digitized by Google

Complements.

79

Write compound subjects before the following predicates: —

May be seen; roar; will be appointed; have flown; has been recommended.

Write compound predicates after the following compound subjects: —

Boys, frogs, and horses; wood, coal, and peat; Maine and New Hampshire; Concord, Lexington, and Bunker Hill; pins, tacks, and needles.

Write compound subjects before the following compound predicates: —

Throb and ache; were tried, condemned, and hanged; eat, sleep, and dress.

Choose your own material and write five sentences, each having a compound subject and a compound predicate.

LESSON 39.

COMPLEMENTS.

Hints for Oral Instruction. — When we say, "The sun *gives*," we express no complete thought. The subject *sun* is complete, but the predicate *gives* does not make a complete assertion. When we say, "The sun *gives light*," we do utter a complete thought. The predicate *gives* is completed by the word *light*. Whatever fills out, or completes, we call a **Complement**. We will therefore call *light* the

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

80

Graded Lessons In English.

complement of the predicate. As *light* completes the predicate by naming the thing acted upon, we call it the **Object Complement**.

Expressions like the following may be written on the board, and by a series of questions the pupils may be made to dwell upon these facts till they are thoroughly understood:—

The officer **arrested** ———; the boy **found** ———;
Charles **saw** ———; coopers **make** ———.

Besides verbs requiring object complements, there are those that do not make complete sense without the aid of a complement of another kind.

A complete predicate does the asserting and expresses what is asserted. In the sentence, "Armies *march*," *march* is a complete predicate, for it does the asserting and expresses what is asserted; viz., marching. In the phrase, *armies marching*, *marching* expresses the act denoted by *march*, but it asserts nothing. In the sentence, "Chalk *is white*," *is* does the asserting, but it does not express what is asserted. We do not wish to assert merely that chalk is or exists. What we wish to assert of chalk is the quality expressed by the adjective *white*. As *white* expresses a quality or attribute, we may call it an **Attribute Complement**.

Using expressions like the following, let the facts given above be drawn from the class by means of questions:—

Grass **growing**; grass **grows**; green **grass**; grass **is green**.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Complements.

81

DEFINITION.—The **Object Complement of a sentence** completes the predicate, and names that which receives the act.

DEFINITION.—The **Attribute Complement of a sentence** completes the predicate and belongs to the subject.

The complement with all its modifiers is called the **Modified Complement**.

ANALYSIS AND PARSING.

Model.—*Fulton invented the first steamboat.*



Explanation of the Diagram.—You will see that the line standing for the object complement is a continuation of the predicate line, and that the little vertical line only touches this without cutting it.

Oral Analysis.—*Fulton* and *invented*, as before. *Steamboat* is the object complement, because it completes the predicate, and names that which receives the act. *The* and *first*, as before. *The first steamboat* is the modified complement.

1. Cæsar crossed the Rubicon.
2. Morse invented the telegraph.
3. Ericsson built the Monitor.
4. Hume wrote a history.
5. Morn purples the east.
6. Antony beheaded Cicero.

Model.—*Gold is malleable.*



Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

82

Graded Lessons in English.

In this diagram, the line standing for the attribute complement, like the object line, is a continuation of the predicate line; but notice the difference in the little mark separating the incomplete¹ predicate from the complement.

Oral Analysis.—*Gold* and *is*, as before. *Malleable* is the attribute complement, because it completes the predicate, and expresses a quality belonging to gold.

7. Pure water is tasteless.
8. The hare is timid.
9. Fawns are graceful.
10. This peach is delicious.
11. He was extremely prodigal.
12. The valley of the Mississippi is very fertile.

LESSON 40.

ERRORS IN THE USE OF MODIFIERS.

Caution.—Place adverbs where there can be no doubt as to the words they modify.

Correct these errors:—

I only bring forward a few things.
Hath the Lord only² spoken by Moses?
We merely speak of numbers.
The Chinese chiefly live upon rice.

¹ Hereafter we shall call the verb the predicate; but, when followed by a complement, it must be regarded as an incomplete predicate.

² Adverbs sometimes modify phrases

Caution. — In placing the adverb, regard must be had to the sound of the sentence.

Correct these errors: —

We always should do our duty.
The times have changed surely.
The work will be never finished.
He must have certainly been sick.

Caution. — Adverbs must not be used for adjectives.

Correct these errors: —

I feel badly.
Marble feels coldly.
She looks nicely.
It was sold cheaply.
It appears still more plainly.
That sounds harshly.
I arrived at home safely.

Caution. — Adjectives must not be used for adverbs.

Correct these errors: —

The bells ring merry.
The curtain hangs graceful.
That is a decided weak point.
Speak no coarser than usual.
These are the words nearest connected.
Talk slow and distinct.
She is a remarkable pretty girl.

It is often difficult to distinguish an adjective complement from an adverb modifier. We offer the following assistance: —

"Mary arrived *safe*." As we here wish to tell the condition of Mary on her arrival, and not the manner of her arriving, we use *safe*, not *safely*. "My head feels *bad*" (is in a bad condition, as perceived by the sense of feeling). "The sun shines *bright*" (is bright — quality — as perceived by its shining).

You must determine whether you wish to tell the quality of the thing named or the manner of the action.

When the idea of **being** is prominent in the verb, as in the examples above, you **see** that the adjective, and not the adverb, follows.

Show that the following adjectives and adverbs are used correctly: —

1. I feel sad.
2. I feel deeply.
3. I feel miserable.
4. He appeared prompt and willing.
5. He appeared promptly and willingly.
6. She looks beautiful.
7. She sings beautifully.

REVIEW QUESTIONS.

What is a conjunction? What is an interjection? Give two rules for the use of the comma (Lesson 37). What is the rule for writing abbreviations? What is the rule for the exclamation point? What is an object complement? What is an attribute complement? Illustrate both. What are the cautions for the position of the adverb? What are the cautions for the use of the adverb and the adjective? Tell when we use the adjective and when we use the adverb.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

COMPOSITION OF SENTENCES AND OF PARAGRAPHS.

SELECTION FROM HABBERTON — "HELEN'S BABIES."

The whistles completed, I was marched with music to the place where the "Jacks" grew. It was just such a place as boys delight in — low, damp, and boggy, with a brook hidden away under overhanging ferns and grasses.

1. The children knew by sight the plant that bore the "Jacks," and every discovery was announced by a piercing shriek of delight. 2. At first I looked hurriedly toward the brook as each yell clove the air; but, as I became accustomed to it, my attention was diverted by some exquisite ferns. 3. Suddenly, however, a succession of shrieks announced that something was wrong, and across a large fern I saw a small face in a great deal of agony. 4. Budge was hurrying to the relief of his brother, and was soon as deeply embedded as Toddie was in the rich, black mud at the bottom of the brook. 5. I dashed to the rescue, stood astride the brook, and offered a hand to each boy, when a treacherous tuft of grass gave way, and, with a glorious splash, I went in myself.

This accident turned Toddie's sorrow to laughter, but I can't say I made light of my misfortune on that account. To fall into clear water is not pleasant, even when one is trout-fishing; but to be clad in white trousers and suddenly drop nearly knee-deep into the lap of mother earth is quite a different thing.

I hastily picked up the children and threw them upon the bank, and then strode off and tried to shake myself, as I have seen a Newfoundland dog do. The shake was not a success — it caused my trousers' legs to flap dismally about my ankles, and sent the streams of treacherous ooze trickling down into my shoes. My hat, of drab felt, had fallen off by the brookside, and been plentifully spattered as I got out.

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"

86

Graded Lessons in English.



"I RUSHED TO THE RESCUE."

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

The Uses of Words and Groups of Words. — We will put the first paragraph above into single sentences.

1. The whistles completed, we were marched with music to the place. 2. The "Jacks" grew in this place. 3. It was a place low, damp, and boggy, with a brook hidden away under overhanging ferns and grasses. 4. Boys delight in such a place.

Find the subject noun (or pronoun) and the predicate verb in each of the four sentences above. Does *the whistles completed* make complete sense? You learned in Lesson 16 that some forms of the verb do not assert — cannot be predicates. Does *brook hidden*, in 3, contain a predicate? What can you say of *hidden*? Find a noun in 3 used to complete the predicate and make the meaning of the subject plainer. What group of adjectives modifies *place*? Tell why these three adjectives are separated by commas. What long phrase describes *place*?

Find the first verb in the second paragraph of the selection. What is the object complement of this verb? *That bore the "Jacks"* does what? The pronoun *that* stands for *plant*. *The plant bore the "Jacks,"* standing by itself, is a complete sentence; but by using *that* for *plant* the whole expression is made to do the work of an adjective. What conjunction joins on another expression that by itself would make a complete sentence? What are the subject and the predicate of this added sentence? *By a piercing shriek of delight* does what? Of what use are the phrases *at first* and *toward the brook* in sentence 2? What group of words is joined to *looked* to tell on what occasion or how often? Find in this group a subject, a predicate, and an object complement. What connects this group to *looked*? What two sentences does *but* here bring together? Does the semicolon show that this connection is close? Point out what you think to be the leading subject and the leading verb after *but*. *By some exquisite ferns* is joined to what? What group of words

goes with *was diverted* to tell when? Find in this group a subject, a predicate, and an attribute complement. Point out in the first part of 3 the leading subject and its verb. What does *suddenly* go with? What does *of shrieks* modify? *However* is loosely thrown in to carry the attention back to what goes before. Notice the commas. Answer the question made by putting *what* after *announced*. In this group of words used as object complement can you find a subject, a predicate, and a complement? What two sentences does *and* here bring together? Point out the subject, the predicate, and the complement in the second of these. *Across a large fern* is joined like an adverb to what? *In a great deal of agony* modifies what? Find a compound predicate in 4. What phrase is joined to *was embedded* to tell where? The group of words *as deeply as Toddie was (embedded)* is joined to what? Find in 5 a compound predicate made up of three verbs, one of which has an object complement.

To the Teacher. — See suggestions with the preceding selection. If our exercises on the second paragraph above are found too hard, the compound and complex sentences may be broken up into single statements.

The Narrative. — This selection from "Helen's Babies" is a story and therefore a narrative. But there are some descriptive touches in it. All stories must have such touches. Perhaps it is not always essential to distinguish between narration and description, but it is worth your while to do it occasionally. Try to point out the descriptive parts in these paragraphs. You certainly can find a descriptive sentence in the first paragraph, and descriptive words, phrases, and clauses throughout the selection. What help to the narrative do these descriptive touches give?

The Paragraphs. — What have you learned about the sentences that make up one paragraph? Are the paragraphs more, or less, closely related than the sentences of each paragraph? Why? Examine these paragraphs and see whether any sentences can be changed

from one paragraph to another. If you think they can, give your reason. Is the order of these paragraphs the right one? Can the order anywhere be changed without throwing the story out of joint? Why?

The General Topic and the Sub-topics.— We shall find that every composition has its general subject, and that each paragraph in the composition has its own particular subject. Let us call the subject of the whole composition the **general topic**. *Sub* means *under*, and so let us call the point which each paragraph develops a **sub-topic**. In the story above we may find some such outline as the following:—

AN EXCURSION IN SEARCH OF "JACKS."

1. The Place where Jacks grow.
2. The Mishap to the Excursionists.
3. The Uncle takes his Seriously.
4. His Attempt at Repairs.

Do you think that such a **framework** helps a writer to tell his story? Do you not think that each sub-topic must suggest some thoughts that the general topic alone would not suggest? If you keep clearly before you the sub-topic of your paragraph, what effect do you think it will have on the thoughts and the sentences of that paragraph? With a good framework before you, must not your story move along in an orderly way from a beginning to an end? Have you ever heard stories badly told? If so, what were the faults?

ORIGINAL COMPOSITION.

Have you not had some experience that you can work up into a good story? If you have, tell the story upon paper, making use of the instruction we have given you in our talk above.

To the Teacher.—Perhaps a reproduction of the story above may be profitable.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

90

Graded Lessons In English.

LESSON 41.

THE POSITION AND USE OF MODIFIERS.

Caution. — Phrase modifiers should be placed as near as may be to the words they modify.

Copy the following, and note the arrangement and the punctuation of the phrases: —

- (g) This place is endeared to me by many associations.
- (h) To me, this place is endeared by many associations.
- (i) Your answers, with few exceptions, have been correctly given.
- (j) He applied for the position, without a recommendation.

When two or more phrases belong to the same word, the one most closely modifying it stands nearest to it.

In the first sentence above, *to me* tells to whom the place is endeared; *by many associations* tells how it is endeared to me, and is therefore placed after *to me*. Try the effect of placing *to me* last. Phrases, like adjectives, may be of different rank.

Notice that *to me*, in (h) above, is transposed, and thus made emphatic, and that it is set off by the comma.

In (i), the phrase is loosely thrown in as if it were not essential, thus making a break in the sentence. To make this apparent to the eye we set the phrase off by the comma.

Place the phrase of (i) in other positions, and set it off. When the phrase is at the beginning or at the end of the sentence, how many commas do you need to set it off? How many, when it is in the middle?

Digitized by Google

Do you find any choice in the four positions of this phrase? After having been told that your answers were correct, would it be a disappointment to be told that they are not all correct? Is the interest in a story best kept up by first telling the important points and then the unimportant particulars? What then do you think of placing this phrase at the end?

What does the last phrase of (*j*) modify? Take out the comma, and then see whether there can be any doubt as to what the phrase modifies.

In the placing of adverbs and phrases great freedom is often allowable, and the determining of their best possible position affords an almost unlimited opportunity for the exercise of taste and judgment.

Such questions as those on (*i*) above may suggest a mode of easy approach to what is usually relegated to rhetoric. Let the pupils see that **phrases** may be **transposed** for various reasons—for **emphasis**, as in (*h*) above; for the purpose of exciting the reader's curiosity and **holding** his **attention** till the complete statement is made, as in (*i*) above, or in, "In the dead of night, with a chosen band, under the cover of a truce, he approached"; and for the sake of **balancing the sentence** by letting some of the modifying terms precede, and some follow, the principal parts; as, "In 1837, on the death of William IV., Victoria succeeded to the throne."

Pupils may note the transposed words and phrases in

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

92

Graded Lessons in English.

the following sentences, and explain their office and the effect of the transposition: —

1. Victories, indeed, they were.
2. Down came the masts.
3. Here stands the man.
4. Doubtful seemed the battle.
5. Wide open stood the door.
6. A mighty man is he.
7. That gale I well remember.
8. Behind her rode Lalla Rookh.
9. Blood-red became the sun.
10. Louder waxed the applause.
11. Him the Almighty Power hurled headlong.
12. Slowly and sadly we laid him down.
13. Into the valley of death rode the six hundred.
14. So died the great Columbus of the skies.
15. Æneas did, from the flames of Troy, upon his shoulders, the old Anchises bear.
16. Such a heart in the breast of my people beats.
17. The great fire up the deep and wide chimney roared.
18. Ease and grace in writing are, of all the acquisitions made in school, the most difficult and valuable.

Read the following sentences in the transposed order, and explain the effect of the change: —

19. He could not avoid it.
20. He would not escape.
21. I must go.
22. He ended his tale here.
23. It stands written so.
24. She seemed young and sad.
25. I will make one more effort to save you.
26. My regrets were bitter and unavailing.
27. I came into the world helpless.
28. A sincere word was never utterly lost.
29. Catiline shall no longer plot her ruin.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

The Position and Use of Modifiers.

93

ORDER OF WORDS IN QUESTIONS.

30. Who wrote the Declaration of Independence?
31. What states border on the Gulf of Mexico?
32. Whom did you see?
33. What is poetry?
34. Which course will you choose?
35. Why are the days shorter in winter?
36. When was America discovered?
37. Were you there?
38. Has the North Pole been reached?

When the interrogative word is subject or a modifier of it, is the order natural, or transposed? See 30 and 31 above.

When the interrogative word is object or attribute complement, or a modifier of either, what is the order? See 32, 33, and 34.

When the interrogative word is an adverb, what is the order? See 35 and 36.

When there is no interrogative word, what is the order? See 37 and 38.

Correct these errors: —

A fellow was arrested with short hair.

He died and went to his rest in New York.

He is to speak of the landing of the Pilgrims at the Academy of Music.

Report any inattention of the waiters to the cashier.

Some garments were made for the family of thick material

The vessel was beautifully painted with a tall mast.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

94

Graded Lessons in English.

I perceived that it had been scoured with half an eye.
A house was built by a mason of brown stone.
A pearl was found by a sailor in a shell.

Punctuate these sentences when corrected.

Caution. — Care must be taken to select the right preposition. For it, consult the Unabridged Dictionaries.

Correct these errors: —

They halted with the river on their backs.
The cat jumped on the chair.
He fell onto the floor.
He went in the house.
Between each page.
He died for thirst.
This is different to that.
The choice lies among the three candidates.
I am angry at him.

Caution. — Do not use two negative, or denying, words so that one shall contradict the other, unless you wish to affirm.

Correct these errors: —

I haven't no umbrella.

Correct by dropping either the adjective *no* or the adverb *not*; as, I have *no* umbrella, or I have *not* an umbrella.

I didn't say nothing.
I can't do this in no way.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Analysis and Parsing.

95

No other emperor was so wise nor powerful.
Nothing can never be annihilated.

LESSON 42.

ANALYSIS AND PARSING

1. Brutus stabbed Cæsar.
2. Man is an animal.
3. Washington captured Cornwallis.
4. Wellington defeated Napoleon at Waterloo.
5. Balboa discovered the Pacific ocean.
6. Vulcan was a blacksmith.
7. The summer has been very rainy.
8. Columbus made four voyages to the New World.
9. The moon reflects the light of the sun.
10. The first vice president of the United States was John Adams.
11. Roger Williams was the founder of Rhode Island.
12. Harvey discovered the circulation of blood.
13. Diamonds are combustible.
14. Napoleon died a prisoner, at St. Helena.
15. In 1619 the first shipload of slaves was landed at Jamestown.

The pupil will notice that *animal*, in sentence 2, is an attribute complement, though it is not an adjective expressing a quality belonging to man, but a noun denoting his class. **Nouns** then may be **attribute complements**.

The pupil will notice also that some of the object and attribute complements above have phrase modifiers.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

96

Graded Lessons In English.

LESSON 43.

COMPOSITION.

Using the following predicates, construct sentences having subjects, predicates, and object complements with or without modifiers: —

— climb — ; — hunt — ; — command — ; —
attacked — ; — pursued — ; — shall receive — ; —
have seen — ; — love — .

Change the following expressions into sentences by asserting the qualities here assumed. Use these verbs for predicates: —

Is, were, appears, may be, became, was, have been, should have been, is becoming, are.

Model. — *Heavy* gold ; Gold *is heavy*.

Green fields ; sweet oranges ; interesting story ; brilliant sunrise ; severe punishment ; playful kittens ; warm weather ; pitiful sight ; sour grapes ; amusing anecdote.

Prefix to the following nouns several adjectives expressing assumed qualities, and then make complete sentences by asserting the same qualities: —

Model. —	white	} chalk.	Chalk <i>is white</i> .
	brittle		Chalk <i>is brittle</i> .
	soft		Chalk <i>is soft</i> .

Gold, pears, pens, lead, water, moon, vase, rock, lakes, summer, ocean, valley.

Digitized by Google

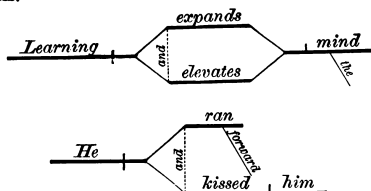
Find your own material, and build two sentences having object complements, and two having attribute complements.

LESSON 44.

ANALYSIS AND PARSING.

MISCELLANEOUS.

Models. —



Explanation of the Diagram. — In the first diagram, the two lines standing for the two parts of the predicate are brought together, and are followed by the complement line. This shows that the two verbs are completed by the same object.

In the second diagram, one of the predicate lines is followed by a complement line; but the two predicate lines are not united, for the two verbs have not a common object.

1. Learning expands and elevates the mind.
2. He ran forward and kissed him.
3. The earth and the moon are planets.
4. The Swiss scenery is picturesque.
5. Jefferson was chosen the third president of the United States

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"

98

Graded Lessons in English.

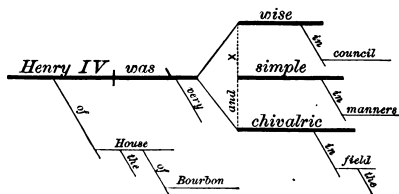
6. Nathan Hale died a martyr to liberty.
7. The man stood speechless.
8. Labor disgraces no man.
9. Aristotle and Plato were the most distinguished philosophers of antiquity.
10. Josephus wrote a history of the Jews.
11. This man seems the leader of the whole party.
12. The attribute complement completes the predicate and belongs to the subject.
13. Lord Cornwallis became governor of Bengal after his disastrous defeat.
14. The multitude ran before him and strewed branches in the way.
15. Peter Minuits traded with the Indians, and bought the whole island of Manhattan for twenty-four dollars.

Pick out the phrases (1) of place and (2) of time in these sentences.

LESSON 45.

ANALYSIS AND PARSING.

MISCELLANEOUS.



Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Explanation of the Diagram. — In this diagram the complement line separates into three parts, to each of which is joined a phrase diagram. The line standing for the word-modifier is joined to that part of the complement line which represents the entire attribute complement.

1. Henry IV., of the House of Bourbon, was very wise in council, simple in manners, and chivalric in the field.
2. Cæsar defeated Pompey at Pharsalia.
3. The diamond is the most valuable gem.
4. The Greeks took Troy by stratagem.
5. The submarine cable unites the continent of America and the Old World.
6. The Gauls joined the army of Hannibal.
7. Columbus crossed the Atlantic with ninety men, and landed at San Salvador.
8. Vulcan made arms for Achilles.
9. Cromwell gained at Naseby a most decisive victory over the Royalists.
10. Columbus was a native of Genoa.
11. God tempers the wind to the shorn lamb.
12. The morning hour has gold in its mouth.
13. The mill of the gods grinds late, but grinds to powder.
14. A young farmer recently bought a yoke of oxen, six cows, and a horse.
15. America has furnished to the world tobacco, the potato, and Indian corn.

Pick out the place and the manner phrases in these sentences. What phrases can you turn into adjectives or adverbs, and what adjectives and adverbs into phrases.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

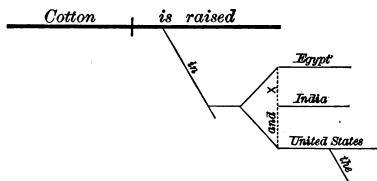
100

Graded Lessons in English.

LESSON 46.

ANALYSIS AND PARSING.

MISCELLANEOUS.



Explanation of the Diagram.— In this diagram the line representing the principal part of the phrase separates into three lines. This shows that the principal part of the phrase is compound. *Egypt*, *India*, and *United States* are all introduced by the same preposition *in*, and have the same relation to *is raised*.

1. Cotton is raised in Egypt, India, and the United States.
2. The navy of Hiram brought gold from Ophir.
3. The career of Cromwell was short.
4. Most mountain ranges run parallel with the coast.
5. Now swiftly glides the bonny boat.
6. An able but dishonest judge presided.
7. The queen bee lays eggs in cells of three different sizes.
8. Umbrellas were introduced into England from China.
9. The first permanent English settlement in America was made at Jamestown, in 1607.
10. The spirit of true religion is social, kind, and cheerful.
11. The summits of the Alps are covered with perpetual snow.

Digitized by Google

12. The months of July and August were named after Julius Caesar and Augustus Caesar.

13. All the kings of Egypt are called, in Scripture, Pharaoh.

14. The bamboo furnishes to the natives of China, shade, food, houses, weapons, and clothing.

Notice that, in 8, *were introduced* is modified by the two phrases *into England* and *from China*. The whole phrase *into England from China* is, then, a **compound phrase**.

Notice that, in 14, *natives*, the principal word of the phrase *to the natives*, is modified by another phrase, *of China*. The whole phrase *to the natives of China* is therefore a **complex phrase**.

Is there another compound or complex phrase in these fourteen sentences? Is there one in the fifteen sentences of Lesson 45?

LESSON 47.

COMPOSITION.

Supply attribute complements to the following expressions. See Caution, Lesson 40.

The marble feels ——. Mary looks ——. The weather continues ——. The apple tastes ——. That lady appears ——. The sky grows ——. The leaves of roses are ——. The undertaking was pronounced ——.

Write a subject and a predicate for each of the following nouns taken as attribute complements: —

Model. — *Soldier*. — That old man has been a *soldier*.

Plant, insect, mineral, vegetable, liquid, gas, solid, historian, poet, artist, traveler, emperor.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

102

Graded Lessons in English.

Using the following nouns as subjects, build sentences each having a simple predicate and two or more object complements:—

Congress, storm, education, king, tiger, hunter, Arnold, shoe-makers, lawyers, merchant.

Build three sentences on each of the following subjects, two of which shall contain object complements, and the third, an attribute complement:—

Model. — *Sun.* — The sun gives light.
The sun warms the earth.
The sun is a luminous body.

Moon, oak, fire, whisky.

LESSON 48.

SUBJECT OR COMPLEMENT MODIFIED BY A PARTICIPLE.

Hints for Oral Instruction. — You have learned, in the preceding lessons, that a quality may be assumed as belonging to a thing; as, *white chalk*, or that it may be asserted of it; as, "*Chalk is white.*" An action, also, may be assumed as belonging to something; as, *Peter turning*, or it may be asserted; as, "*Peter turned.*" In the expression, "*Peter, turning, said,*" which word expresses an action as assumed, and which asserts an action? Each pupil may give an example of an action asserted and of an action assumed; as, "*Corn grows,*" *corn growing*; "*Geese gabble,*" *geese gabbling.*

Digitized by Google

Subject or Complement modified by a Participle. 103

This form of the verb, which merely assumes the act, being, or state, is called a **participle**.

When the words *growing* and *gabbling* are placed before the nouns, thus: *growing corn*, *gabbling geese*, they tell simply the kind of corn and the kind of geese, and are therefore adjectives.

When *the* or some other adjective is placed before these words, and a preposition after them, thus: *The growing of* the corn, *the gabbling of* the geese, they are simply the names of actions, and are therefore nouns.

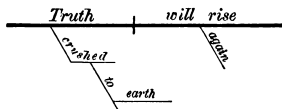
Let each pupil give an example of a verb asserting an action, and change it to express:—

- (1) An assumed action; (2) A permanent quality;
- (3) The name of an action.

Participles may be completed by objects and attributes.

ANALYSIS AND PARSING.

Model. — *Truth, crushed to earth, will rise again.*



Explanation of the Diagram. — In this diagram, the line standing for the principal word of the participial phrase is broken; one part slants, and the other is horizontal. This shows that the participle *crushed* is used like an adjective to modify *Truth*, and yet retains the nature of a verb, expressing an action received by truth.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

104

Graded Lessons in English.

Oral Analysis. — This is a sentence, because — ; *Truth* is the subject, because — ; *will rise* is the predicate, because — ; the phrase, *crushed to earth*, is a modifier of the subject, because — ; *crushed* introduces the phrase and is the principal word in it ; the phrase *to earth* is a modifier of *crushed* ; *to* introduces it, and *earth* is the principal word in it ; *again* is a modifier of the predicate, because — . *Truth crushed to earth* is the modified subject, *will rise again* is the modified predicate.

Parsing. — *Crushed* is the form of the verb called participle. The action expressed by it is merely assumed.

1. The mirth of Addison is genial, imparting a mild glow of thought.
2. The general, riding to the front, led the attack.
3. The balloon, shooting swiftly into the clouds, was soon lost to sight.
4. Wealth acquired dishonestly will prove a curse.
5. The sun, rising, dispelled the mists.
6. The thief, being detected, surrendered to the officer.
7. They boarded the vessel lying in the harbor.
8. The territory claimed by the Dutch was called New Netherlands.
9. Washington, having crossed the Delaware, attacked the Hessians stationed at Trenton.
10. Burgoyne, having been surrounded at Saratoga, surrendered to General Gates.
11. Pocahontas was married to a young Englishman named John Rolfe.
12. A shrug of the shoulders, translated into words, loses much force.
13. The armies of England, mustered for the battles of Europe, do not awaken sincere admiration.

Digitized by Google

Note that the participle, like the predicate verb, may consist of two or more words.

Note, too, that the participle, like the adjective, may belong to a noun complement.

LESSON 49.

THE INFINITIVE PHRASE.

Hints for Oral Instruction.— There is another form of the verb which, like the participle, cannot be the predicate of a sentence, for it cannot assert; as, "She went out to see a friend;" "*To lie* is a disgrace." As this form of the verb expresses the action, being, or state in a general manner, without limiting it directly to a subject, it is called an **Infinitive**, which means without limit. The infinitive generally follows *to*; as, *to walk*, *to sleep*.

Let each pupil give an infinitive.

The infinitive and the preposition *to* constitute an **Infinitive phrase**, which may be employed in several ways.

T.—"I have a duty *to perform*." The infinitive phrase modifies what? P.—The noun *duty*. T.—It then performs the office of what? P.—Of an adjective modifier.

T.—"I come *to hear*." The infinitive phrase modifies what? P.—The verb *come*. T.—What office then does it perform? P.—That of an adverb modifier.

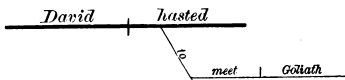
T.—"To lie is base." What is base? **P.**—To lie.
T.—"He attempted to speak." What did he attempt?
P.—To speak. **T.**—"To lie is a subject, and to speak is an object. What part of speech is used as subject and object? **P.**—The noun.

T.—The **Infinitive phrase** is used as an **adjective**, an **adverb**, and a **noun**.

Infinitives may be completed by objects and attributes.

ANALYSIS AND PARSING.

Model.—*David hastened to meet Goliath.*

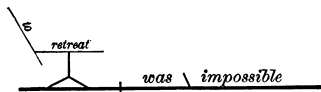


Analysis of the Infinitive Phrase.—*To* introduces the phrase; *meet*, completed by the object *Goliath*, is the principal part.

Parsing of the Phrase.—*To* is a preposition, because —; *meet* is a verb, because —; *Goliath* is a noun, because —.

1. I come not here to talk.
2. I rejoice to hear it.
3. A desire to excel leads to eminence.
4. Dr. Franklin was sent to France to solicit aid for the colonies.
5. To retreat was impossible.

To here merely introduces the infinitive phrase.



The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Position and Punctuation of Participial Phrase. 107

Explanation of the Diagram.—As this phrase subject cannot, in its proper form, be written on the subject line, it is placed above, and, by means of a support, the phrase diagram is made to rest on the subject line. The phrase complement may be diagramed in a similar way, and made to rest on the complement line.

6. The hands refuse to labor.
7. To live is not all of life.
8. The Puritans desired to obtain religious freedom.
9. The Romans, having conquered the world, were unable to conquer themselves.
10. Narvaez sailed from Cuba to conquer Florida.
11. Some savages of America and Africa love to wear rings in the nose.
12. Andrew Jackson, elected to succeed J. Q. Adams, was inaugurated in 1829.

LESSON 50.

POSITION AND PUNCTUATION OF THE PARTICIPIAL PHRASE.

See Lesson 37, and Caution 1 in Lesson 41. Correct these sentences, and punctuate them when corrected:—

A house was built for a clergyman having seven gables.
The old man struck the saucy boy raising a gold-headed cane.
We saw a marble bust of Sir W. Scott entering the vestibule.
Here is news from a neighbor boiled down.
I found a cent walking over the bridge.
Balboa discovered the Pacific ocean climbing to the top of a mountain.

Punctuate the following exercises:—

Cradled in the camp Napoleon was the darling of the army.
Having approved of the plan the king put it into execution.

Digitized by Google.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

108

Graded Lessons in English.

Satan incensed with indignation stood unterrified.
My friend seeing me in need offered his services.
James being weary with his journey sat down on the wall.
The owl hid in the tree hooted through the night.

REVIEW QUESTIONS.

What binds together the sentences of a paragraph? What is the general topic? What are the sub-topics? What is a framework? How is it formed? Of what help would it be to the writer?

Give the Caution relating to the position of the phrase modifier. If phrases are of different rank, which should stand nearest to the word they modify? Illustrate. Why are phrases transposed? Illustrate their transposition for various reasons. What sentences of 1-19, Lesson 41, contain transposed prepositional phrases? What ones contain transposed adjectives? Adverbs? Nouns or pronouns used as objects? As attribute complements? What did you transpose in 19-29? What is transposed in sentences that ask questions? Give the Caution relating to the choice of prepositions. That relating to double negatives. Give examples of errors. What is a compound phrase? A complex? What is a participle? When may it become an adjective? It may be completed by what? What is an infinitive? An infinitive phrase? What offices may such a phrase perform? Illustrate. The infinitive may be completed by what? The *to* of an infinitive phrase may sometimes do what only?

COMPOSITION OF SENTENCES AND OF PARAGRAPHS.

SELECTION FROM GEORGE ELIOT.

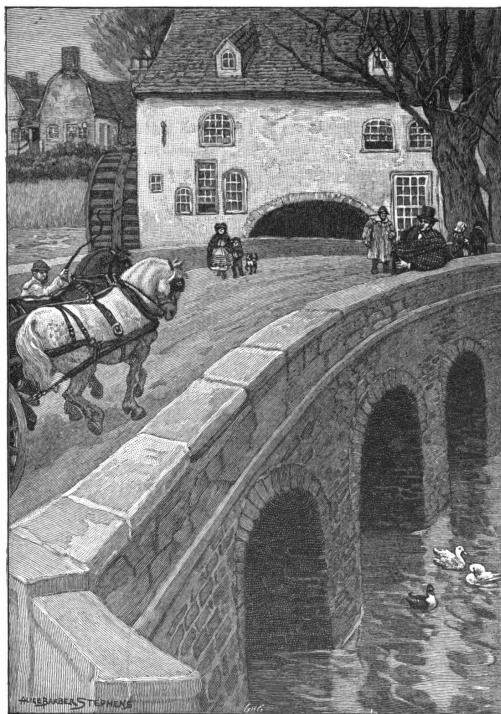
And this is Dorlcote Mill. I must stand a minute or two here on the bridge and look at it, though the clouds are threatening and it is far on in the afternoon. Even in this leafless time of departing February, it is pleasant to look at. Perhaps the chill, damp season adds

Digitized by Google

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"

Composition.

109



"DORLCOTE MILL."

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

110

Graded Lessons in English.

a charm to the trimly kept building, as old as the elms and chestnuts that shelter it from the northern blast.

The stream is brimful now, and half drowns the grassy fringe in front of the house. As I look at the stream, the vivid grass, the delicate, bright-green softening the outline of the great trunks and branches that gleam from under the bare purple boughs, I am in love with moistness, and envy the white ducks that are dipping their heads far into the water, unmindful of the awkward appearance they make in the drier world above.

1. And now there is the huge covered wagon, coming home with sacks of grain. 2. That honest wagoner is thinking of his dinner, which is getting sadly dry in the oven at this late hour; but he will not touch it till he has fed his horses — the strong, submissive beasts, who, I fancy, are looking mild reproach at him from between their blinkers, that he should crack his whip at them in that awful manner, as if they needed such a hint! 3. See how they stretch their shoulders up the slope toward the bridge, with all the more energy because they are so near home. 4. Look at their grand, shaggy feet, that seem to grasp the firm earth, at the patient strength of their necks bowed under the heavy collar, at the mighty muscles of their struggling haunches. 5. I should like to see them, with their moist necks freed from the harness, dipping their eager nostrils into the pond.

The Uses of Words and Groups of Words. — Notice that in sentence 1, third paragraph, the subject is placed after the predicate. Tell what *now* and *there* do. *Coming home with sacks of grain* does what? Does *coming* express action? Does it assert action? What is it? What does *home* do? Put *its* before *home* and then read the whole phrase. What other change do you find necessary? A noun is sometimes used alone to do the work of an adverb phrase, the preposition being omitted. What is the office of *minute* in the second sentence of the first paragraph? What preposition could be

Digitized by Google

put in? In 2, third paragraph, the pronoun *which* stands for *dinner*. Read the sentence, using the noun instead of the pronoun. Have you now two sentences, or one? You see that *which* not only stands for *dinner*, but it joins on a sentence so as to make it describe the dinner. What does *till he has fed his horses* do? Omitting *till*, would this group of words be a sentence? What, then, joins this group, and makes it do the work of an adverb? Notice the dash after *horses*. The writer here breaks off rather suddenly and begins again, using *beasts* instead of *horses*. To *beasts* are added many descriptive words. You will learn that this noun *beasts* added to the noun *horses* is called an explanatory modifier. Notice that *I fancy* is thrown in loosely or independently and is set off by commas. All the other words beginning with *who* and ending with *hint* are joined by *who* to *beasts*. Notice that the writer makes these beasts think like persons, and so uses *who* instead of *which* or *that*. Do we ordinarily speak of looking anything? In *who are looking reproach*, what is the object complement of *are looking*? What long group of words made up of two sentences tells why the beasts are looking reproach? Read separately the main divisions of 2. What conjunction connects these? Is one of these divisions itself divided into parts by commas? Should then some mark of wider separation be put between the main divisions of 2? To build so long a sentence as 2 is venturesome. We advise young writers not to make such attempts. It is hard to write very long sentences and keep the meaning clear. In 3 the subject of *see* is *you*, which is generally omitted in a command. You are here told to see what? Break this long object complement up into two sentences. What do the horses stretch? Where do they stretch their shoulders? How do they stretch? Why do they stretch with more energy? What is the subject of *look* in 4? The phrase beginning with *at* and ending with *earth* does what? Find two other long phrases introduced by *at* and tell what they do.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

112

Graded Lessons in English.

That seem to grasp the firm earth goes with what? Put the noun *feet* in place of the pronoun *that* and make a separate sentence of this group. What word, then, makes an adjective modifier of this sentence and joins it to *feet*? Does *to grasp* assert action? What do you call it? It is here used as attribute complement. *Bowed under the heavy collar* describes what? Does *bowed* assert action? What do you call it?

To the Teacher. — If time permits, such exercises as the above may profitably be continued. See suggestions with preceding exercises.

Descriptive Writing. — This extract from the novelist who called herself "George Eliot" we have slightly changed for our purpose. It is purely **descriptive**. It is a painting in words — a vivid picture of a very pretty scene. How grateful we are to those who can, as it were, turn a page of a book into canvas, and paint on it a rich verbal picture that delights us every time we read it or recall it! How many such pictures there are in our libraries! And how little they cost us when compared with those that we buy and hang upon our walls!

Some Features of a Good Description. — Does this author mention many features of the mill, of the stream, and of the horses pulling their load over the bridge? Do those that she does mention suggest to you everything else? Name some of the things suggested to you but not mentioned in this description. Does not some of the charm of a description lie in the reader's having something left him to supply? If the author had given you every little detail of the mill, the stream, and the laboring horses, would not the description have been dull and tiresome? What things that the author imagined but did not really see are mentioned in the third paragraph? Do these touches of fancy or imagination help the picture? Do they show that the author was in love with her work? and do they therefore stimulate your fancy or imagination?

Digitized by Google

The Framework. — In making a framework for this description would you take for the general topic "The Scene from the Bridge" or "Things Seen from a Bridge"? or would you prefer some other wording of it? Now write out a framework, placing the sub-topics under the general topic as you have been taught.

ORIGINAL COMPOSITION.

Describe some scene that you greatly enjoy, or draw your picture from imagination. Make a framework and try to profit by all that we have said.

LESSON 51.

REVIEW.

Correct these miscellaneous errors. See Cautions in Lessons 30, 40, and 41 :—

There never was such another man.
He was an old venerable patriarch.
John has a cadaverous, hungry, and lean look.
He was a well-proportioned, fine fellow.
Pass me them potatoes.
Put your trust not in money.
We have often occasion for thanksgiving.
Now this is to be done how?
Nothing can justify ever profanity.
To continually study is impossible.

An adverb is seldom placed between the preposition *to* and the infinitive.

Mary likes to tastefully dress.
Learn to carefully choose your words.
She looks queerly.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

114

Graded Lessons in English.

Give me a soon and direct answer.

The post stood firmly.

The eagle flies highly.

The orange tastes sweetly.

I feel tolerable well.

The branch breaks easy.

Thistles grow rapid.

The eagle flies swift.

This is a miserable poor pen.

A wealthy gentleman will adopt a little boy with a small family.

A gentleman called from Africa to pay his compliments.

Water consists in oxygen and hydrogen.

He went out attended with a servant.

I have a dislike to such tricksters.

We have no prejudice to foreigners.

She don't know nothing about it.

Father wouldn't give me none.

He hasn't been sick neither.

I won't have no more nohow.

To the Teacher. — See that a good reason is given for every correction.

LESSON 52.

COMPOSITION.

Build sentences in which the following participles shall be used as modifiers: —

Being fatigued ; laughing ; being amused ; having been elected ;
running ; having been running.

Digitized by Google

Expand each of the following sentences into three sentences, using the participial form of the verb as a participle in the first; the same form as an adjective in the second; and as a noun in the third : —

Model. — The stream *flows*; The stream, *flowing* gently, crept through the meadow; The *flowing* stream slipped away to the sea; The *flowing* of the stream caused a low murmur.

The stream flows. The sun rises. Insects hum. The birds sing. The wind whistles. The bells are ringing. The tide ebbs.

Form infinitive phrases from the following verbs, and use these phrases as adjectives, adverbs, and nouns, in sentences of your own : —

Smoke, dance, burn, eat, lie, try.

LESSON 53.

NOUNS AND PRONOUNS AS MODIFIERS.

Hints for Oral Instruction. — In the sentence, "The *robin's eggs* are blue," the noun *robin's* does what? **P.** — It tells what or whose eggs are blue. **T.** — What word names the things owned or possessed? **P.** — *Eggs*. **T.** — What word names the owner or possessor? **P.** — *Robin's*.

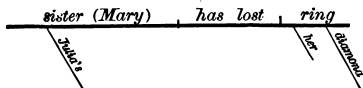
T. — The noun *robin's* is here used as a modifier. You see that this word, which I have written on the board, is the word *robin* with a little mark (') called an apostrophe, and the letter *s* added. These are added to denote possession.

In the sentence, "*Webster, the statesman*, was born in New Hampshire," the noun *statesman* modifies the subject *Webster* by explaining what or which Webster is meant. Both words name the same person.

Let the pupils give examples of each of these two kinds of **Noun Modifiers** — the **Possessive** and the **Explanatory**.

ANALYSIS AND PARSING.

Model. — *Julia's sister Mary has lost her diamond ring.*

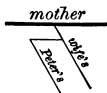


Explanation of the Diagram. — *Mary* is written on the subject line, because *Mary* and *sister* both name the same person, but the word *Mary* is inclosed within marks of parenthesis to show that *sister* is the proper grammatical subject.

In **oral analysis**, call *Julia's* and *Mary* modifiers of the subject, *sister*, because *Julia's* tells whose sister, and *Mary* explains *sister* by adding another name of the same person. *Her* is a modifier of the object, because it tells whose ring is meant.

Julia's sister Mary is the modified subject, the predicate is unmodified, and *her diamond ring* is the modified object complement.

1. The planet Jupiter has four moons.
2. The Emperor Nero was a cruel tyrant.
3. Peter's wife's mother lay sick of a fever.



4. An ostrich outruns an Arab's horse.
5. His pretty little nephew Arthur had the best claim to the throne.
6. Milton, the great English poet, became blind.
7. Cæsar gave his daughter Julia in marriage to Pompey.
8. London, the capital of England, is the largest and richest city in the world.
9. Joseph, Jacob's favorite son, was sold by his brethren to the Ishmaelites.
10. Alexander the Great¹ was educated under the celebrated philosopher, Aristotle.
11. Friends tie their purses with a spider's thread.
12. Cæsar married Cornelia, the daughter of Cinna.
13. His fate, alas! was deplorable.
14. Love rules his kingdom without a sword.

LESSON 54.

COMPOSITION.

Nouns and pronouns denoting possession may generally be changed to equivalent phrases; as, *Arnold's treason* = *the treason of Arnold*. Here the preposition *of* indicates possession, the relation expressed by the apostrophe (') and s. Change the following possessive nouns to equivalent phrases, and the phrases indicating possession to possessive nouns, and then expand the expressions into complete sentences:—

¹ *Alexander the Great* may be taken as one name, or *Great* may be called an explanatory modifier of *Alexander*.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

118

Graded Lessons In English.

Model. — The *earth's* surface ; the surface of *the earth* is made up of land and water.

The earth's surface ; Solomon's temple ; England's King ; Washington's Farewell Address ; Dr. Kane's Explorations ; Peter's wife's mother ; George's friend's father ; Shakespeare's plays ; Noah's dove ; the diameter of the earth ; the daughter of Jephthah ; the invasion of Burgoyne ; the voyage of Cabot ; the Armada of Philip ; the attraction of the earth ; the light of the moon.

Find for the things mentioned below, other names which shall describe or explain them. Add such names to these nouns, and then expand the expressions into complete sentences : —

Model. — Ink. — *Ink, a dark fluid,* is used in writing.

Observe the following rule : —

COMMA — RULE. — An **Explanatory Modifier**, when it does not restrict the modified term or combine closely with it, is set off by the comma.

New York, rain, paper, the monkey, the robin, tea, Abraham Lincoln, Alexander Hamilton, world, peninsula, Cuba, Shakespeare.

The chief difficulty in the punctuation of the different kinds of modifiers is in determining whether or not they are restrictive. The following examples illustrate the difficulty : —

- (a) The words *golden* and *orion* are pleasant to the ear.
- (b) Words, the signs of ideas, are spoken and written.
- (c) Use words that are current.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Composition.

119

- (d) Words, which are the signs of ideas, are spoken and written.
- (e) The country anciently called Gaul is now called France.
- (f) France, anciently called Gaul, derived its name from the Franks.
- (g) Glass bends easily when it is hot.
- (h) I met him in Paris, when I was last abroad.

In (a) the application of *words* is limited, or restricted, to the two words mentioned; in (c) *words* is **restricted to a certain kind**. In (b) and (d) the modifiers **do not restrict**. They apply to all words and simply **add information**. In (e) the participial phrase restricts the application of *country* to one particular country; but in (f) the phrase describes without limiting. The omission of the comma in (g) shows that "*Glass bends easily*" is not offered as a general statement, but that the action is restricted to a certain time or condition. *When it is hot* is essential to the intended meaning. The punctuation of (h) shows that the speaker does not wish to make the time of meeting a prominent or essential part of what he has to say. The adverb clause simply gives additional information. If (h) were an answer to the question, When did you meet him? the comma would be omitted. The sense may be varied by the use or the omission of the comma.

Let the pupils see how incomplete the statements are when the restrictive modifiers are omitted, and that the other modifiers are not so necessary to the sense, but are supplementary. In such expressions as *I myself, we boys,*

Digitized by Google

the explanatory words are not restrictive, but they combine closely with the modified term.

Write three sentences, each of which shall contain a noun or pronoun denoting possession, and a noun or pronoun used to explain.

Study these possessive forms with reference to the possessive signs: —

The sailor's story; the farmer's son; the pony's mane; the monkey's tail; a day's work; James's book; a cent's worth; a man's wages; the child's toys; the woman's hat; the sailors' stories; the farmers' sons; the ponies' manes; the monkeys' tails; three days' work; five cents' worth; two men's wages; those children's toys; women's hats.

Pick out the nouns above that are in the singular and tell which of the two possessive signs — ('s) or (') — they add. Which do nouns in the plural add? Note, above, three exceptions in the plural. *Man*, *woman*, and *child* do not add the regular *s*-ending to form their plurals. The plurals are *men*, *women*, and *children*, and the possessive form of these has the full ('s) — as is seen above.

Study these possessive forms: —

Dombey & Son's business; J. J. Little & Co.'s printing-house; William the Conqueror's reign; Reed and Kellogg's series of grammars are Maynard, Merrill, & Company's publications.

When a group of words is treated as a compound name, which word of the group takes the possessive sign?

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Analysis and Parsing.

121

LESSON 55.

ANALYSIS AND PARSING.

MISCELLANEOUS EXAMPLES IN REVIEW.

1. The toad spends the winter in a dormant state.
2. Pride in dress or in beauty betrays a weak mind.
3. The city of London is situated on the river Thames.
4. Napoleon Bonaparte was born in 1769, on an island in the Mediterranean.
5. Men's opinions vary with their interests.
6. Ammonia is found in the sap of trees, and in the juices of all vegetables.
7. Earth sends up her perpetual hymn of praise to the Creator.
8. Having once been deceived by him, I never trusted him again.
9. *Æsop*, the author of *Æsop's Fables*, was a slave.
10. Hope comes with smiles to cheer the hour of pain.
11. Clouds are collections of vapors in the air.
12. To relieve the wretched was his pride.
13. Greece, the most noted country of antiquity, scarcely exceeded in size the half of the state of New York.

What phrases above are (1) adjectival, (2) adverbial, (3) compound, and (4) complex? What compound phrase has a complex as a part of it?

LESSON 56.

ANALYSIS AND PARSING.

MISCELLANEOUS EXAMPLES IN REVIEW—CONTINUED.

1. We are never too old to learn.
2. Civility is the result of good nature and good sense.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

122

Graded Lessons In English.

3. The right of the people to instruct their representatives is generally admitted.
4. The immense quantity of matter in the Universe presents a most striking display of Almighty power.
5. Virtue, diligence, and industry, joined with good temper and prudence, must ever be the surest means of prosperity.
6. The people called Quakers were a source of much trouble to the Puritans.
7. The Mayflower brought to America¹ one hundred and one men, women, and children.
8. Edward Wingfield, an avaricious and unprincipled man, was the first president of the Jamestown colony.
9. John Cabot and his son Sebastian, sailing under a commission from Henry VII. of England, discovered the continent of America.
10. True worth is modest and retiring.
11. Jonah, the prophet, preached to the inhabitants of Nineveh.

In the exercise you take for pleasure, in the errands on which you are sent, and in your going to and from school, I will suppose that, like the girl in the picture, you walk ; or, like the boy in the picture, you ride on your wheel.

We are all reasonable beings. This means that, whether or not we think of it at the time, we always have a reason for everything we do. If we walk, we do it for reasons that seem good to us ; if we ride, we do it for other reasons that seem good to us.

Now, without suggestions from any one, think, if possible, of at least four reasons, satisfactory to you, why

¹ One hundred and one may be taken as one adjective.



A GIRL ON FOOT AND A BOY SPINNING BY ON A WHEEL.

you go about on foot, if you do ; why you go about on a wheel, if you do—reasons which you can urge in defence and justification of your method of locomotion. These reasons will form an **Argument**.

Then under the heading, or topic,

WHY I GO ABOUT ON FOOT,

OR

WHY I GO ABOUT ON A WHEEL,

array, one after another, in the most natural order you can think of, these good reasons, each beginning, "I walk

because," etc. ; or "I ride on a wheel because," etc. Then expand each reason into a paragraph.

These reasons, or **sub-topics**, in proper order, numbered, and standing under the **general topic**, form an **outline** or **framework**. The development of each sub-topic forms, as we have seen, a **paragraph**, which is made up of sentences in proper order. The relative **length** of the paragraphs depends upon the relative importance of the sub-topics ; and the paragraphs grouped together constitute a **composition**, or **theme**.

LESSON 57.

COMPLEX SENTENCES.

THE ADJECTIVE CLAUSE.

Hints for Oral Instruction.—A word-modifier may sometimes be expanded into a phrase or into an expression that asserts.

T.—"A *wise* man will be honored." Expand *wise* into a phrase, and give me the sentence. **P.**—"A man *of wisdom* will be honored." **T.**—Expand *wise* into an expression that asserts, join this to *man*, as a modifier, and then give me the entire sentence. **P.**—"A man *who is wise* will be honored."

T.—You see that the same quality may be expressed in three ways—A *wise* man, A man *of wisdom*, A man *who is wise*.

Let the pupils give similar examples.

T. — In the sentence, "A man *who is wise* will be honored," the word *who* stands for what? P. — For the noun *man*. T. — Then what part of speech is it? P. — A pronoun.

T. — Put the noun *man* in the place of the pronoun *who*, and then give me the sentence. P. — "*A man, man is wise*, will be honored."

T. — I will repeat your sentence, changing the order of the words — "*A man will be honored.*" "*Man is wise.*" Is the last sentence now joined to the first as a modifier, or are they two separate sentences? P. — They are two separate sentences.

T. — Then you see that the pronoun *who* not only stands for the noun *man*, but it connects the modifying expression, *who is wise*, to *man*, the subject of the sentence, "*A man will be honored*," and thus there is formed what we call a **Complex Sentence**. These two parts we call **Clauses**. "*A man will be honored*" is the **Independent Clause**; *who is wise* is the **Dependent Clause**.

Clauses that modify nouns or pronouns are called **Adjective Clauses**.

DEFINITION. — A **Clause** is a part of a sentence containing a subject and its predicate.

DEFINITION. — A **Dependent Clause** is one used as an adjective, an adverb, or a noun.

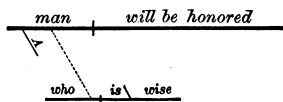
DEFINITION. — An **Independent Clause** is one not dependent on another clause.

DEFINITION.—A **Simple Sentence** is one that contains but one subject and one predicate, either or both of which may be compound.

DEFINITION.—A **Complex Sentence** is one composed of an independent clause and one or more dependent clauses.

ANALYSIS AND PARSING.

Model.—



Explanation of the Diagram.—You will notice that the lines standing for the subject and predicate of the independent clause are heavier than those of the dependent clause. This pictures to you the relative importance of the two clauses. You will see that the pronoun *who* is written on the subject line of the dependent clause. But this word performs the office of a conjunction also, and this office is expressed in the diagram by a dotted line. As all modifiers are joined by slanting lines to the words they modify, you learn from this diagram that *who is wise* is a modifier of *man*.

Oral Analysis.—This is a complex sentence, because it consists of an independent clause and a dependent clause. "*A man will be honored*" is the independent clause; *who is wise* is the dependent clause. *Man* is the subject of the independent clause; *will be honored* is the predicate. The word *A* and the clause, *who is wise*, are modifiers of the subject. *A* points out *man*, and *who is wise* tells the kind of man. *A man who is wise* is the modified subject; the predicate is unmodified. *Who* is the subject of the dependent clause, *is* is the predicate, and *wise* is the attribute complement. *Who* connects the two clauses.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Composition.

127

1. He that runs may read.
2. Man is the only animal that laughs and weeps.
3. Henry Hudson discovered the river which bears his name.
4. He necessarily remains weak who never tries exertion.
5. The meridians are those lines that extend from pole to pole.
6. He who will not be ruled by the rudder must be ruled by the rock.
7. Animals that have a backbone are called vertebrates.
8. Uneasy lies the head that wears a crown.
9. The thick mists which prevail in the neighborhood of Newfoundland are caused by the warm waters of the Gulf Stream.
10. The power which brings a pin to the ground holds the earth in its orbit.
11. Death is the black camel which kneels at every man's gate.
12. Our best friends are they who tell us of our faults, and help us to mend them.

The pupil will notice that, in some of these sentences, the dependent clause modifies the subject, and that, in others, it modifies the noun complement.

COMMA — RULE. — The **adjective** or the **adverb clause**, when it does not closely follow and restrict the word modified, is generally set off by the comma.

LESSON 58.

COMPOSITION.

ADJECTIVE CLAUSES.

Expand each of the following adjectives into

- (1) A phrase, (2) A clause.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

128

Graded Lessons in English.

and then use these three modifiers in three separate sentences of your own construction: —

Model. — *Energetic; of energy;* $\left\{ \begin{array}{l} \textit{who has energy,} \\ \textit{or} \\ \textit{who is energetic.} \end{array} \right.$

An *energetic* man will succeed; a man *of energy* will succeed; a man *who has energy* (or *who is energetic*) will succeed.

Honest, long-eared, beautiful, wealthy.

Expand each of the following possessive nouns into

(1) A phrase, (2) A clause,

and then use these three modifiers in three separate sentences: —

Model. — *Saturn's rings*; the rings *of Saturn*; the rings *which surround Saturn*.

Saturn's rings can be seen with a telescope; the rings *of Saturn* can be seen with a telescope; the rings *which surround Saturn* can be seen with a telescope.

Absalom's hair; the hen's eggs; the elephant's tusks.

Change the following simple sentences into complex sentences by expanding the participial phrases into clauses.

The vessels carrying the blood from the heart are called arteries.

The book prized above all other books is the Bible.

Rivers rising west of the Rocky Mts. flow into the Pacific ocean.

The guns fired at Concord were heard around the world.

Digitized by Google

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"

Complex Sentences.

129

LESSON 59.

COMPLEX SENTENCES.

THE ADVERB CLAUSE.

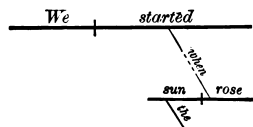
Hints for Oral Instruction. — You learned in Lesson 33 that an adverb can be expanded into an equivalent phrase ; as, "The book was *carefully* read" = "The book was read *with care*."

You are now to learn that a phrase used as an adverb may be expanded into an **Adverb clause**. In the sentence, "We started *at sunrise*," what phrase is used like an adverb? P. — *At sunrise*. T. — Expand this phrase into an equivalent clause, and give me the entire sentence. P. — "We started *when the sun rose*."

T. — You see that the phrase, *at sunrise*, and the clause, *when the sun rose*, both modify *started*, telling the time of starting, and are therefore equivalent to adverbs. We will then call such clauses **Adverb clauses**.

ANALYSIS AND PARSING.

Model —



Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

130

Graded Lessons in English.

Explanation of the Diagram. — The line which connects the two predicate lines pictures three things. It is made up of three parts. The upper part shows that *when* modifies *started*; the lower part, that it modifies *rose*; and the dotted part shows that it connects.

Oral Analysis. — This is a complex sentence, because — ; *We started* is the independent clause, and *when the sun rose* is the dependent clause. *We* is the subject of the independent clause, and *started* is the predicate. The clause, *when the sun rose*, is a modifier of the predicate, because it tells when we started. *Started when the sun rose* is the modified predicate.

Sun is the subject of the dependent clause, and *rose* is the predicate, and *the* is a modifier of *sun*; *the sun* is the modified subject. *When* modifies *rose* and *started*, and connects the clause-modifier to the predicate *started*.

Parsing of *when*. — *When* is an adverb modifying the two verbs *started* and *rose*, and thus connects the two clauses. It modifies these verbs by showing that the two actions took place at the same time.

1. The dew glitters when the sun shines.
2. Printing was unknown when Homer wrote the Iliad.
3. Where the bee sucks honey, the spider sucks poison.
4. Ah! few shall part where many meet.
5. Where the devil cannot come, he will send.
6. While the bridegroom tarried, they all slumbered and slept.
7. Fools rush in where angels fear to tread.
8. When the tale of bricks is doubled, Moses comes.
9. When I look upon the tombs of the great, every emotion of envy dies within me.
10. The upright man speaks as he thinks.
11. He died as the fool dieth.
12. The scepter shall not depart from Judah until Shiloh come.

Digitized by Google

When, *while*, and *until* indicate time; the clauses they introduce are **adverb clauses of time**. Select them. *Where* indicates place; the clauses it introduces are **adverb clauses of place**. How many are there in the sentences above? *As* here indicates manner. Pick out the **adverb clauses of manner** which it introduces. In

The ground is wet *because* it rains; *if* the night is cloudy, no dew will fall; *though* it rained during the night, the ground is now dry; Nature puts us to sleep at night *that* she may repair the waste of the body by day,

the clauses introduced by (1) *because*, (2) *if*, (3) *though*, and (4) *that* are **adverb clauses of** (1) **cause**, (2) **condition**, (3) **concession**, and (4) **purpose**. The words introducing them are not conjunctive adverbs but pure conjunctions, and stand where *when* stands, in the diagram above, but on a line dotted throughout.

See if, in the offices which these common adverb clauses perform, you can find the justification of their names.

LESSON 60.

COMPOSITION.

ADVERB CLAUSES.

Expand each of the following phrases into an adverb clause, and fit this clause into a sentence of your own:—

Model. — *At sunset*; *when the sun set*. We returned *when the sun set*.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

132

Graded Lessons in English.

At the hour ; on the playground ; by moonlight ; in youth ; among icebergs ; after school ; at the forks of the road ; during the day ; before church ; with my friend.

To each of the following independent clauses, join an adverb clause, and so make complex sentences : —

— Peter began to sink. The man dies —. Grass grows —. Iron — can easily be shaped. The rattlesnake shakes his rattle —. — a nation mourns. Pittsburg stands —. He dared to lead —.

An adverb clause may stand before the independent clause, between its parts, or after it ; as, "*When it is hot*, glass bends easily" ; "Glass, *when it is hot*, bends easily" ; "Glass bends easily *when it is hot*." Notice the punctuation of these examples.

Adverb clauses may be contracted in various ways. Clauses introduced by the comparatives *as* and *than* are usually found in an abbreviated form ; as, "You are as old *as* he (*is old*)" ; "You are older *than* I (*am old*)."
Attention may be called to the danger of mistaking here the nominative for the objective. We suggest making selections for the study of adverb clauses.

REVIEW QUESTIONS.

In what two ways may nouns be used as modifiers ? Illustrate. Nouns and pronouns denoting possession may sometimes be changed into what ? Illustrate. Give the rule for the punctuation of explanatory modifiers. Into what may an adjective be expanded ? Into what may a participial phrase be expanded ? Give illustrations.

Digitized by Google

Give an example of a complex sentence. Of a clause. Of an independent clause. Of a dependent clause. Into what may a phrase used as an adverb be expanded? Illustrate the seven classes of adverb clauses spoken of in Lesson 59.

COMPOSITION OF SENTENCES AND OF PARAGRAPHS.

SELECTION FROM THE BROTHERS GRIMM.

Once upon a time there was a very old man, whose eyes were dim, whose ears were dull, and whose knees trembled. When he sat at



table, he could scarcely hold his spoon ; and often he spilled his food over the tablecloth and sometimes down his clothes.

His son and daughter-in-law were much vexed about this, and at last they made the old man sit behind the oven in a corner, and gave

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

134

Graded Lessons In English.

him his food in an earthen dish, and not enough of it either; so that the poor man grew sad, and his eyes were wet with tears. Once his hand trembled so much that he could not hold the dish, and it fell upon the ground and broke all in pieces, so that the young wife scolded him; but he made no reply and only sighed. Then they bought him a wooden dish, and out of that he had to feed.

One day, as he was sitting in his usual place, he saw his little grandson, four years old, fitting together some pieces of wood. "What are you making?" asked the old man.

"I am making a wooden trough," replied the child, "for father and mother to feed out of when I grow big."

At these words the father looked at his wife for a moment, and presently they began to cry. Henceforth they let the old grandfather sit at table with them, and they did not even say anything if he spilled a little food upon the cloth.

The Uses of Words and Groups of Words.—What is the order of subject and predicate in the first sentence of this selection? The word *there* does not tell where; it is put before *was* to let the subject follow. *There* is frequently so used and is then called an independent a/verb. Find in the first sentence three adjective clauses. What connects each to *man*? What other office has this connective? How are these adjective clauses connected with one another? What is the office of the dependent clause in the next sentence? If this clause were placed after its principal clause, would the comma be needed? Are the clauses separated by the semicolon as closely connected as those divided by the comma?

After *made* and some other words the *to* before the infinitive is omitted. Find such an instance in the first sentence of the second paragraph. In this same sentence change *gave him his food*, making *him* come last. You have learned that a noun or a pronoun may be used without a preposition to do the work of an adverb phrase.

Digitized by Google

What does *one day* do in the third paragraph? Is a preposition needed before *day*? In the same sentence *years* is used adverbially to modify the adjective *old*. It would be hard to find a preposition to put before *years*. We might say "old to the extent of four years," but *four years* answers for the whole phrase. In this same paragraph what words are quoted exactly as the old man uttered them? Notice that the next quotation is broken by the words *replied the child*, and so each part of the quotation is separately inclosed within quotation marks. (See next lesson.)

To the Teacher. — We have here touched a few features of the sentences above. The exercises given with the preceding selections will suggest a fuller examination of the phrases and clauses.

Suggestions from this Narrative. — We see that this beautiful story has a purpose. Its purpose is to teach us kindness to our parents. It is well planned. Every sentence and every paragraph is adapted to the end in view. No useless item or circumstance is admitted. The story stops when the end is reached. Anything added to the fifth paragraph would spoil the story. We certainly can learn much from such a model.

Paragraphs. — Does every sentence in the first paragraph aid in picturing the helplessness of the old grandfather? Is the picture complete? Does the second paragraph strongly impress us with the unkindness of the son and daughter-in-law, who ought to have been moved to pity by the old man's condition? Does it contain an unnecessary sentence? In telling how the grandchild unconsciously taught a lesson, a dialogue is introduced, and so what really belongs to one sub-topic is put in the form of two paragraphs. It is customary to make a separate paragraph of each single speech in a dialogue. Read the last paragraph carefully and see whether one could wish to know anything more about the effect of the lesson taught by the child.

Make a **framework** for this story.

ORIGINAL COMPOSITION.

Make up a short story from your own experience, or from your imagination, and try to profit by the suggestions above. Prepare a framework at the beginning.

LESSON 61.

THE NOUN CLAUSE.

Hints for Oral Instruction.—"That stars are suns" is taught by astronomers." What is taught by astronomers? P.—That stars are suns. T.—What, then, is the subject of *is taught*? P.—The clause, *That stars are suns*. T.—This clause, then, performs the office of what part of speech? P.—Of a noun.

T.—"Astronomers teach *that stars are suns*." What do astronomers teach? P.—That stars are suns. T.—What is the object complement of *teach*? P.—The clause, *that stars are suns*. T.—What office, then, does this clause perform? P.—That of a noun.

T.—"The teaching of astronomers is, *that stars are suns*." What does *is* assert of teaching? P.—That stars are suns. T.—What, then, is the attribute complement? P.—*That stars are suns*. T.—Does this complement express the quality of the subject, or does it name the thing that the subject names? P.—It names the thing that the subject names. T.—It is equivalent then to what part of speech? P.—To a noun.

The Uncertainty Principle

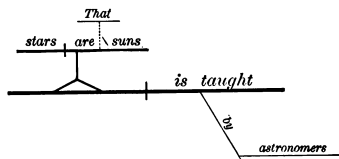
Volume Orange Issue Four "Over the Horizon"

The Noun Clause.

137

T. — You see then that a clause, like a noun, may be used as the subject or the complement of a sentence.

ANALYSIS AND PARSING.



You will understand this diagram from the explanation of the second diagram in Lesson 49.

Oral Analysis. — This is a complex sentence, in which the whole sentence takes the place of the independent clause. *That stars are suns* is the dependent clause. *That stars are suns* is the subject of the whole sentence. *That* simply introduces the dependent clause.

In **parsing**, call *that* a conjunction.

1. That the Scotch are an intelligent people is generally acknowledged.
2. That the moon is made of green cheese is believed by some boys and girls.
3. That Julius Cæsar invaded Britain is a historic fact.
4. That children should obey their parents is a divine precept.
5. I know that my Redeemer liveth.
6. Plato taught that the soul is immortal.
7. Peter denied that he knew his Lord.
8. Mahomet found that the mountain would not move.
9. The principle maintained by the colonies was, that taxation without representation is unjust.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

138

Graded Lessons In English.

10. Our intention is, that this work shall be well done.
11. Our hearts' desire and prayer is, that you may be saved.
12. The belief of the Sadducees was, that there is no resurrection of the dead.

Look at the noun clauses in these sentences :—

1. Goldsmith says, "Learn the luxury of doing good."
2. Goldsmith says that we should learn the luxury of doing good.
3. "The owlet Atheism, hooting at the glorious sun in heaven, cries out, 'Where is it?'"
4. Coleridge compares atheism to an owlet hooting at the sun, and asking where it is.
5. "To read without reflecting," says Burke, "is like eating without digesting."
6. May we not find "sermons in stones and good in everything"?
7. There is much meaning in the following quotation: "Books are embalmed minds."
8. We must ask, What are we living for?
9. We must ask what we are living for.

Notice that the writer of (1) has copied into his sentence (quoted) the exact language of Goldsmith. The two marks, like inverted commas, and the two marks, like apostrophes, which inclose this copied passage (quotation), are called **Quotation Marks**.

Name all the differences between (1) and (2). Is the same thought expressed in both? Which quotation would you call **direct**? Which, **indirect**?

Notice that the whole of (3) is a quotation, and that this quotation contains another quotation inclosed within

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

single marks. Notice the order of the marks at the end of (3).

Point out the differences between (3) and (4). In which is a question quoted just as it would be asked? In which is a question merely referred to? Which question would you call **direct**? Which, **indirect**? Name every difference in the form of these.

In which of the above sentences is a quotation interrupted by a parenthetical clause? How are the parts marked?

Point out a quotation that cannot make complete sense by itself. How does it differ from the others as to punctuation and the first letter?

In (7) a **Colon** precedes the quotation to show that it is introduced in a formal manner by the word *following*.

In (8) a question is introduced without quotation marks. Questions that, like this, are introduced without being referred to any particular person or persons, are often written without quotation marks. State the differences between (8) and (9).

In quoting a question, the interrogation point must stand within the quotation marks; but, when a question contains a quotation, this order is reversed. Point out illustrations above.

Selections written in the colloquial style and containing frequent quotations and questions may be taken from reading books, for examination, discussion, and copying.

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"

140

Graded Lessons in English.

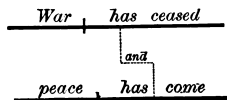
LESSON 62.

COMPOUND SENTENCES.

ANALYSIS AND PARSING.

DEFINITION. — A **Compound Sentence** is one composed of two or more independent clauses.

Model. — *War has ceased, and peace has come.*



Explanation of the Diagram. — These two clause diagrams are shaded alike to show that the two clauses are of the same rank. The connecting line is not slanting, for one clause is not a modifier of the other. As one entire clause is connected with the other, the connecting line is drawn between the predicates simply for convenience.

Oral Analysis. — This is a compound sentence, because it is made up of two independent clauses. The first clause, etc. —.

1. Morning dawns, and the clouds disperse.
2. Prayer leads the heart to God, and he always listens.
3. A soft answer turneth away wrath, but grievous words stir up anger.
4. Power works easily, but fretting is a perpetual confession of weakness.
5. Many meet the gods, but few salute them.
6. We eat to live, but we do not live to eat.
7. The satellites revolve in orbits around the planets, and the planets move in orbits around the sun.

Digitized by Google

Sentences Classified with Respect to Meaning. 141

8. A wise son maketh a glad father, but a foolish son is the heaviness of his mother.

9. Every man desires to live long, but no man would be old.

10. ¹Pride goeth before destruction, and a haughty spirit before a fall.

11. Towers are measured by their shadows, and great men by their calumniators.

12. Worth makes the man, and want of it the fellow.

LESSON 63.

SENTENCES CLASSIFIED WITH RESPECT TO THEIR MEANING

Hints for Oral Instruction.— You have already become acquainted with three kinds of sentences. Can you name them? **P.**— The Simple sentence, the Complex, and the Compound.

T.— These classes have been made with regard to the form of the sentence. We will now arrange sentences in classes with regard to their meaning.

"Mary sings." *"Does Mary sing?"* *"Sing, Mary."* *"How Mary sings!"* Here are four simple sentences. Do they all mean the same thing? **P.**— They do not.

T.— Well, you see they differ. Let me tell you wherein. The first one tells a fact, the second asks a question, the third expresses a command, and the fourth expresses sudden thought or strong feeling. We call the first a **Declara-**

¹ A verb is to be supplied in each of the last three sentences.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

142

Graded Lessons in English.

tive sentence, the second an **Interrogative sentence**, the third an **Imperative sentence**, and the fourth an **Exclamatory sentence**.

DEFINITION. — A **Declarative Sentence** is one that is used to affirm or to deny.

DEFINITION. — An **Interrogative Sentence** is one that expresses a question.

DEFINITION. — An **Imperative Sentence** is one that expresses a command or an entreaty.

DEFINITION. — An **Exclamatory Sentence** is one that expresses sudden thought or strong feeling.

INTERROGATION POINT — RULE. — Every **direct interrogative sentence** should be followed by an interrogation point.

COMPOSITION.

Change each of the following declarative sentences into three interrogative sentences, and tell how the change was made: —

Model. — *Girls can skate; Can girls skate? How can girls skate? What girls can skate?*

You are happy. Parrots can talk. Low houses were built.

Change each of the following into an imperative sentence. Notice that independent words are set off by the comma: —

Model. — *Carlo eats his dinner; Eat your dinner, Carlo.*

George plays the flute. Birdie stands on one leg.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Analysis and Parsing.

143

Change each of the following into exclamatory sentences: —

Model. — *You are happy; How happy you are! What a happy child you are! You are so happy!*

Time flies swiftly. I am glad to see you. A refreshing shower fell. Lapland is a cold country. It is hot between the tropics.

Write a declarative, an interrogative, an imperative, and an exclamatory sentence on each of the following topics: —

Weather, lightning, a stage coach.

LESSON 64.

ANALYSIS AND PARSING.

MISCELLANEOUS EXERCISES IN REVIEW.

In the analysis, classify these sentences first with reference to their form, and then with reference to their meaning: —

1. Wickedness is often made a substitute for wit.
2. Alfred was a brave, pious, and patriotic prince.
3. The throne of Philip trembles while Demosthenes speaks.
4. That the whole is equal to the sum of its parts is an axiom.
5. The lion belongs to the cat tribe, but he cannot climb a tree.
6. Pride is a flower that grows in the devil's garden.
7. Of all forms of habitation, the simplest is the burrow.
8. When the righteous are in authority, the people rejoice.
9. When the wicked beareth rule, the people mourn.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

144

Graded Lessons in English.

10. ¹ Cassius, be not deceived.
11. How rich, how poor, how abject, how august, how wonderful is man !
12. Which is the largest city in the world ?

LESSON 65.

ANALYSIS AND PARSING.

MISCELLANEOUS EXERCISES IN REVIEW—CONTINUED.

1. Politeness is the oil which lubricates the wheels of society.
2. O liberty ! liberty ! how many crimes are committed in thy name !
3. The mind is a goodly field, and to sow it with trifles is the worst husbandry in the world.
4. Every day in thy life is a leaf in thy history.
5. Make hay while the sun shines.
6. Columbus did not know that he had discovered a new continent.
7. The subject of inquiry was, Who invented printing ?
8. The cat's tongue is covered with thousands of little sharp cones, pointing towards the throat.
9. The fly sat upon the axle of a chariot wheel and said, "What a dust do I raise !"
10. Sir Humphrey Gilbert, attempting to recross the Atlantic in his little vessel, the Squirrel, went down in mid-ocean.
11. Charity begins at home, but it should not stay there.
12. The morn, in russet mantle clad, walks o'er the dew of yon high eastern hill.

¹ *Cassius* is independent, and may be diagramed like an interjection. The subject of *be deceived*, is *thou* or *you*, understood.

Digitized by Google



"AT RECESS."

This is a picture of pupils playing games at recess. Some of the games boys alone play; some, girls play; and some, boys and girls together play.

The boys of the class may select at least three of these games, and in as many paragraphs may tell how they are played — what they are; the girls may take the three or more which they like best, and do the same with them.

Confine yourselves to the essential features of each game. See how much good thought upon these points

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

146

Graded Lessons in English.

you can put into the paragraphs. Choose apt and simple words, arrange them with care, and diversify your sentences in kind and in length, making what you have to say clear and strong by your way of saying it.

The composition, telling what some things are, will be **Expository**.

LESSON 66.

MISCELLANEOUS ERRORS IN REVIEW.

I haven't near so much. I only want one. Draw the string tightly. He writes good. I will prosecute him who sticks bills upon this church or any other nuisance. Noah for his godliness and his family were saved from the flood. We were at Europe this summer. You may rely in that. She lives to home. I can't do no work. He will never be no better. They seemed to be nearly dressed alike. I won't never do so no more. A ivory ball. An hundred head of cattle. george washington. gen dix of n y. o sarah i Saw A pretty Bonnet. are You going home? A young man wrote these verses who has long lain in his grave for his own amusement. This house will be kept by the widow of Mr. B. who died recently on an improved plan. (In correcting the position of the adjective clauses in the two examples above, observe the caution for the phrase modifiers, Lesson 41.) He was an independent smail farmer. The mind knows feels and thinks. The urchin was ragged barefooted dirty homeless and friendless. I am some tired. .This here road is rough. That there man is homely. pshaw i am so Disgusted. Whoa can't you stand still. James the gardener gave me a white lily. Irving the genial writer lived on the hudson.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Composition.

147

LESSON 67.

COMPOSITION.

Construct one sentence out of each group of the sentences which follow : —

Model. — An *able* man was chosen.

A *prudent* man was chosen.

An *honorable* man was chosen.

An *able, prudent, and honorable* man was chosen.

Pure water is destitute of color.

Pure water is destitute of taste.

Pure water is destitute of smell.

Cicero was the greatest orator of his age.

Demosthenes was the greatest orator of his age.

Daisies peeped up here.

Daisies peeped up there.

Daisies peeped up everywhere.

Expand each of the following sentences into three : —

The English language is spoken in England, Canada, and the United States. The Missouri, Ohio, and Arkansas rivers are branches of the Mississippi.

Out of the four following sentences compose one sentence having three explanatory modifiers : —

Model. — Elizabeth was *the daughter of Henry VIII.*

Elizabeth was *sister of Queen Mary.*

Elizabeth was *the patron of literature.*

Elizabeth defeated the Armada.

Elizabeth, *the daughter of Henry VIII, sister of Queen Mary, and the patron of literature,* defeated the Armada.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

148

Graded Lessons in English.

Boston is the capital of Massachusetts.
Boston is the Athens of America.
Boston is the "Hub of the Universe."
Boston has crooked streets.

Expand the following sentence into four sentences :—

Daniel Webster, the great jurist, the expounder of the Constitution, and the chief of the "American Triumvirate," died with the words, "I still live," on his lips.

LESSON 66.

COMPOSITION.

Change the following simple sentences into complex sentences by expanding the phrases into adjective clauses :—

Model.— People *living in glass houses* shouldn't throw stones.
People *who live in glass houses* shouldn't throw stones.

Those living in the Arctic regions need much oily food.
A house built upon the rock will stand.
The boy of studious habits will always have his lesson.
Wellington was a man of iron will.
A scholar without money is not a bankrupt.
Everybody has something to teach us.
A race shortening its weapons extends its boundaries.

Change the following complex sentences into simple sentences by contracting the adjective clauses into phrases :—

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Composition

149

Much of the cotton which is raised in the Gulf States is exported.
The house which was built upon the sand fell.
A thing which is beautiful is a joy forever.
Aaron Burr was a man who had fascinating manners.
Glaciers, which flow down mountain gorges, obey the law of rivers.
The best sermon which was ever preached on modern society is
"Vanity Fair."

In mere love of what was vile, Charles II. surpassed all his subjects.
A common English ending is *er*, which is indicative of the agent.

Change the following simple sentences into complex sentences by expanding the phrases into adverb clauses:—

Model.—Birds return *in the spring*; *when spring comes*, the birds return.

The dog came at call. In old age our senses fail.

Shakespeare died at his birthplace.

Wishing to enjoy the Adirondacks, you must carry mountains in your brain.

Staying at home, one may visit Italy and the tropics.

Death, delaying his visits long, will certainly knock at every door.

A shrug of the shoulders, translated into words, would lose much.

Modern failures are of such magnitude as to appall the imagination.

Change the following complex sentences into simple sentences by contracting the adverb clauses into phrases:—

The ship started when the tide was at flood.

When he reached the middle of his speech, he stopped.

Error dies of lockjaw if she scratches her finger.

Some minute animals feed though they have no mouths.

Roads are built that travelers may be accommodated.

Shakespeare died where he was born.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

150

Graded Lessons In English.

Supply noun clauses and make complete sentences out of the following expressions : —

— is a well-known fact. The fact was —. Ben Franklin said —.

Contract the dependent clauses of these sentences into phrases : —

Arnold was fearful that he should be detected.

When one has eaten honey, one's tea seems to be without sugar.

Cairo is situated where the Ohio joins the Mississippi.

The effect of friction is, that it heats the substances rubbed.

He had no place where he might lay his head.

That we should defend ourselves is a duty.

If the farmer allows the weeds to grow unchecked, he will gather no harvest.

Mohammedans promise that they will obey the teachings of the Koran.

Dark clothes are warm in summer, because they absorb the rays of the sun.

LESSON 69.

GENERAL REVIEW.

What is a letter? Give the name and the sound of each of the letters in the three following words : *letters, name, sound*. Into what classes are letters divided? Define each class. Subdivide the consonants. Name the vowels. What is a word? What is verbal language? What is English Grammar? What is a sentence? What is the difference between the two expressions, *ripe apples* and *apples are ripe*? What two parts must every sentence have? Define each.

Digitized by Google

General Review.

151

What is the analysis of a sentence? What is a diagram? What are parts of speech? How many parts of speech are there? Give an example of each. What is a noun? What is a verb? What must every predicate contain? What is a pronoun? What is a modifier? What is an adjective? What adjectives are sometimes called articles? When is *a* used? When is *an* used? Illustrate. Give an example of one modifier joined to another. What is an adverb? What is a phrase? A compound phrase? A complex? What is a preposition? What is a conjunction? What is an interjection? Give four rules for the use of capital letters (Lessons 8, 15, 19, 37). Give two rules for the use of the period, one for the exclamation point, and one for the interrogation point (Lessons 8, 37, 63).

LESSON 70.

GENERAL REVIEW.

What is an object complement? What is an attribute complement? How does a participle differ from a predicate verb? Illustrate. What offices does an infinitive phrase perform? Illustrate. How are sentences classified with respect to form? Give an example of each class. What is a simple sentence? What is a clause? What is a dependent clause? What is an independent clause? What is a complex sentence? What is a compound sentence? How are sentences classified with respect to meaning? Give an example of each class. What is a declarative sentence? What is an interrogative sentence? What is an imperative sentence? What is an exclamatory sentence? What different offices may a noun perform? *Ans.*—A noun may be used as a subject, as an object complement, as an attribute complement, as a possessive modifier, as an explanatory modifier, as the principal word in a prepositional phrase, and it may be used independently. Illustrate

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

152

Graded Lessons in English.

each use. What are sometimes substituted for nouns? *Ans.* — Pronouns, phrases, and clauses. Illustrate. What is the principal office of a verb? What offices may be performed by a phrase? What, by a clause? What different offices may an adjective perform? What parts of a speech may connect clauses? *Ans.* — Conjunctions, adverbs, and pronouns (Lessons 62, 59, and 57). Give rules for the use of the comma (Lessons 37, 54, 57). Give and illustrate the directions for using adjectives and adverbs, for placing phrases, for using prepositions, and for using negatives (Lessons 40, 41).

To the Teacher. — For additional review, see "Scheme," p. 267.

COMPOSITION OF SENTENCES AND OF PARAGRAPHS.

SELECTION FROM BEECHER.

Overwork almost always ends in weakening the digestive organs. There are those who overtax their minds through months and years, forgetful that there is a close connection between overwork and dyspepsia. Every one should remember that there is a point beyond which he cannot urge his brain without harm to his stomach; and that, when he loses his stomach, he loses the very citadel of health. The whole body is renewed from the blood, and the blood is made from the food taken into the stomach. The power of the blood to renew bone and brain and muscle depends upon a good digestion.

Too little sleep is fatal to health. Perhaps you have to work hard all day; but that is no reason why you should resolve, "If I cannot have pleasure by day, I will have it at night." You are taking the very substance of your body when you burn the lamp of pleasure till one or two o'clock in the morning. God has made sleep to be a sponge with which to rub out fatigue. A man's roots are planted in night, as a tree's are planted in soil, and out of it he should come, at waking, with fresh growth and bloom. As a rule, you should take eight hours of the twenty-four, for sleep.

Digitized by Google

The Uses of Words and Groups of Words.—In the exercises under the selection from the Brothers Grimm what did you learn about *there* as used twice in the second sentence above? What does *those* mean? What long adjective clause is joined to *those* by *who*? Does this clause read so closely as not to need a comma before *who*? Does *forgetful* describe the persons represented by *who*? Why is a comma used before *forgetful*? You learned in a preceding exercise that a noun may do the work of an adverb phrase without the help of a preposition. A noun clause may do the same. The adjective *forgetful* is modified by the noun clause, *that . . . dyspepsia*. If we say *forgetful of the fact*, we see that the noun clause means the same as *fact* and has the same office. What two long noun clauses are used to complete *should remember*? What conjunction introduces each of these clauses? What conjunction joins them together? What mark of punctuation between? If one of these noun clauses were not itself divided into clauses by the comma, would the semicolon be needed? The clause, *beyond . . . stomach*, goes with what word? *When . . . stomach* modifies what verb? Classify the sentences of this paragraph as simple, complex, or compound.

To the Teacher.—We have here treated informally some difficult points. Perhaps these may be better understood when the book is reviewed.

The Various Objects Writers Have.—From your study of the preceding selections you learn that a writer may have any one of several objects in writing. He may wish simply to instruct the reader, as does Darwin in what he says of earthworms. He may wish merely to amuse the reader, as does Mr. Habberton in our extract from "Helen's Babies." He may wish only to put before us a picture which, like that of George Eliot's, shall afford delight. Or he may wish to get hold of what we call our wills and lead us to do something, perform some duty. This is what the story from the Brothers Grimm aims at.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

154

Graded Lessons in English.

And you saw how it does this — by working on our feelings. There are at least these four objects that a writer may propose to himself. Which of these four objects has Mr. Beecher in the paragraphs we quote? Does he instruct? Does he try to get us to do something? Would it help you to have clearly before you from the beginning the object you are seeking to accomplish?

Figurative Expressions. — In these paragraphs Mr. Beecher calls a man's stomach the citadel of health, and sleep a sponge to rub out fatigue with, and says a man's roots are planted in night. He does not use these words *citadel*, *sponge*, and *roots* in their first or common meaning. He uses them in what we call a **figurative sense**. He means to say that a man's stomach is to him what a fortress is to soldiers, a source of strength; that in sleep fatigue disappears as do figures on a slate or blackboard when a wet sponge is drawn across them; and that a man gets out of night what a tree's roots draw out of the soil — nourishment and vigor. Such figurative uses of words give strength and beauty to style.

ORIGINAL COMPOSITION.

In the paragraphs quoted above, you were told of the effects on health of overwork and of insufficient sleep. Perhaps you can write of exercise, of proper food, of clothes, or of some other things on which health may depend.

To the Teacher. — If the early presentation of an outline of technical grammar is not compelled by a prescribed course of study, we should here introduce a series of lessons in the construction of sentences, paragraphs, and themes.

Here is an exercise in combining simple statements into complex and compound sentences, and in resolving complex and compound sentences into simple sentences. In combining statements, it is an excellent practice for the pupil to contract, expand, transpose, and to substitute other words. They thus learn to express the same thought in a variety of ways. Any reading-book or history will furnish good material for such practice.

Digitized by Google

Combine in as many ways as possible each of the following groups of sentences : —

Example. — This man is to be pitied. He has no friends.

1. This man has no friends, and he is to be pitied.
2. This man is to be pitied, because he has no friends.
3. Because this man has no friends, he is to be pitied.
4. This man, who has no friends, is to be pitied.
5. This man, having no friends, is to be pitied.
6. This man, without friends, is to be pitied.
7. This friendless man deserves our pity.

1. The ostrich is unable to fly. It has not wings in proportion to its body.

2. Egypt is a fertile country. It is annually inundated by the Nile.

3. The nerves are little threads, or fibers. They extend from the brain. They spread over the whole body.

4. John Gutenberg published a book. It was the first book known to have been printed on a printing-press. He was aided by the patronage of John Faust. He published it in 1455. He published it in the city of Mentz.

5. The human body is a machine. A watch is delicately constructed. This machine is more delicately constructed. A steam-engine is complicated. This machine is more complicated. A steam-engine is wonderful. This machine is more wonderful.

You see that short sentences closely related in meaning may be improved by being combined. But young writers frequently use too many *ands* and other connectives, and make their sentences too long.

Long sentences should be broken up into short ones when the relations of the parts are not clear.

As clauses may be joined to form sentences, so, as you have learned, sentences may be united to make paragraphs.

The first word of a paragraph should, as you have seen, begin a new line, and should be written a little farther to the right than the first words of other lines.

Combine the following statements into sentences and paragraphs, and make of them a complete composition, or theme:—

Water is a liquid. It is composed of oxygen and hydrogen. It covers about three-fourths of the surface of the earth. It takes the form of ice. It takes the form of snow. It takes the form of vapor. The air is constantly taking up water from rivers, lakes, oceans, and from damp ground. Cool air contains moisture. Heated air contains more moisture. Heated air becomes lighter. It rises. It becomes cool. The moisture is condensed into fine particles. Clouds are formed. They float across the sky. The little particles unite and form raindrops. They sprinkle the dry fields. At night the grass and flowers become cool. The air is not so cool. The warm air touches the grass and flowers. It is chilled. It loses a part of its moisture. Drops of dew are formed. Water has many uses. Men and animals drink it. Trees and plants drink it. They drink it by means of their leaves and roots. Water is a great purifier. It cleanses our bodies. It washes our clothes. It washes the dust from the leaves and the flowers. Water is a great worker. It floats vessels. It turns the wheels of mills. It is converted into steam. It is harnessed to mighty engines. It does the work of thousands of men and horses.

To the Teacher.—Condensed statements of facts, taken from some book not in the hands of your pupils, may be read to them, and they may be required to expand and combine these and group them into paragraphs.

PARTS OF SPEECH SUBDIVIDED.

LESSON 71.

CLASSES OF NOUNS.

Hints for Oral Instruction. — Hereafter, in the "Hints," we shall drop the dialogue form, but we expect the teacher to continue it. A poor teacher does all the talking, a good teacher makes the pupils talk.

The teacher may here refer to his talk about the classification of birds, and show that, after birds have been arranged in great classes, such as robins, sparrows, etc., these classes will need to be subdivided if the pupil is to be made thoroughly acquainted with this department of the animal kingdom. So, after grouping words into the eight great classes, called Parts of Speech, these classes may be divided into other classes. For instance, take the two nouns *city* and *Brooklyn*. The word *city* is the common name of all places of a certain class, but the word *Brooklyn* is the proper or particular name of an individual of this class. We have here then two kinds of nouns which we call **Common** and **Proper**.

Let the teacher write a number of nouns on the board,

and require the pupil to classify them and give the reasons for the classification.

To prepare the pupil thoroughly for this work, the teacher will find it necessary to explain why such words as *music*, *mathematics*, *knowledge*, etc. are common nouns. *Music*, e.g., is not a proper noun, for it is not a name given to an individual thing to distinguish it from other things of the same class. There are no other things of the same class—it forms a class by itself. So we call the noun *music* a common noun.

CLASSES OF PRONOUNS.

The speaker seldom refers to himself by name, but uses the pronoun *I* instead. In speaking to a person, we often use the pronoun *you* instead of his name. In speaking of a person or thing that has been mentioned before, we say *he* or *she* or *it*. These words that by their form indicate the speaker, the hearer, or the person or thing spoken of are called **Personal Pronouns** (Lesson 19).

Give sentences containing nouns repeated, and require the pupils to improve these sentences by substituting pronouns.

When we wish to refer to an object that has been mentioned in another clause, and at the same time to connect the clauses, we use a class of pronouns called **Relative Pronouns**. Let the teacher illustrate by using the pronouns *who*, *which*, and *that* (Lesson 57).

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Classes of Nouns.

159

When we wish to ask about anything whose name is unknown, we use a class of pronouns called **Interrogative Pronouns**. The interrogative pronoun stands for the unknown name and asks for it ; as, "*Who* comes here?" "*What* is this?"

"*Both men* were wrong." Let us omit *men* and say, "*Both* were wrong." You see the meaning is not changed — *both* is here equivalent to *both men*, that is, it performs the office of an adjective and that of a noun. It is therefore an **Adjective Pronoun**. Let the teacher further illustrate the office of the adjective pronoun by using the words *each, all, many, some, such*, etc.

DEFINITIONS.

CLASSES OF NOUNS.

A **Common Noun** is a name which belongs to all things of a class.

A **Proper Noun** is the particular name of an individual.

CLASSES OF PRONOUNS.

A **Personal Pronoun** is a pronoun that by its form denotes the speaker, the one spoken to, or the one spoken of.

A **Relative Pronoun** is one that relates to some preceding word or words and connects clauses.

An **Interrogative Pronoun** is one with which a question is asked.

An **Adjective Pronoun** is one that performs the offices of both an adjective and a noun.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

160

Graded Lessons in English.

LESSON 72.

COMPOSITION.

Build each of the following groups of nouns into a sentence. See Rule, Lesson 15.

webster cares office washington repose home marshfield.

george washington commander army revolution president united states westmoreland state virginia month february.

san francisco city port pacific trade united states lines steamships sandwich islands japan china australia.

Write five simple sentences, each containing one of the five personal pronouns: *I, thou* or *you, he, she*, and *it*.

Write four complex sentences, each containing one of the four relative pronouns: *who, which, that*, and *what*.

What is used as a relative pronoun when the antecedent is omitted. The word for which a pronoun stands is called its **Antecedent**. When we express the antecedent, we use *which* or *that*. "I shall do *what* is required;" "I shall do the *thing which* is required, or *that* is required."

Build three interrogative sentences, each containing one of the three interrogative pronouns *who, which*, and *what*.

Build eight sentences, each containing one of the adjective pronouns *few, many, much, some, this, these, that, those*.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Classes of Adjectives.

161

LESSON 73.

CLASSES OF ADJECTIVES.

Hints for Oral Instruction. — When I say *large, round, sweet, yellow oranges*, the words *large, round, sweet, and yellow* modify the word *oranges* by telling the kind, and limit the application of the word to oranges of that kind.

When I say *this orange, yonder orange, one orange*, the words *this, yonder, and one* do not tell the kind, but simply point out or number the orange, and limit the application of the word to the orange pointed out or numbered.

Adjectives of the first class describe by giving a quality, and so are called **Descriptive Adjectives**.

Adjectives of the second class define by pointing out or numbering, and so are called **Definitive Adjectives**.

Let the teacher write nouns on the board, and require the pupils to modify them by appropriate descriptive and definitive adjectives.

DEFINITIONS.

A **Descriptive Adjective** is one that modifies by expressing quality.

A **Definitive Adjective** is one that modifies by pointing out, numbering, or denoting quantity.

COMPOSITION.

Place the following adjectives in two columns, one headed descriptive and the other definitive, then build simple sentences in which they shall be employed as modi-

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

162

Graded Lessons In English.

fiers. Find out the meaning of each word before you use it:—

Round, frolicsome, first, industrious, jolly, idle, skillful, each, the, faithful, an, kind, one, tall, ancient, modern, dancing, mischievous, stationary, nimble, several, slanting, parallel, oval, every.

Build simple sentences in which the following descriptive adjectives shall be employed as attribute complements. Let some of these attributes be compound:—

Restless, impulsive, dense, rare, gritty, sluggish, dingy, selfish, clear, cold, sparkling, slender, graceful, hungry, friendless.

Build simple sentences in which the following descriptive adjectives shall be employed. Some of these adjectives have the form of participles, and others are derived from proper nouns:—

Shining, moving, swaying, bubbling, American, German, French, Swiss, Irish, Chinese.

CAPITAL LETTER—RULE.—An **Adjective** derived from a proper noun must begin with a capital letter.

LESSON 74.

CLASSES OF VERBS.

Hints for Oral Instruction.—“The man *caught*” makes no complete assertion and is not a sentence. If we add the object complement *fish*, we complete the assertion and

Digitized by Google

form a sentence — "The man *caught* fish." The action expressed by *caught* passes over from the man to the fish. *Transitive* means passing over, and hence all those verbs that express an action that passes over from the doer to something which receives are called **Transitive Verbs**.

"Fish *swim*." The verb *swim* does not require an object to complete the sentence. No action passes from a doer to a receiver. These verbs which express action that does not pass over to a receiver, and all those which do not express action at all, but simply being or state of being, are called **Intransitive Verbs**.

Let the teacher write transitive and intransitive verbs on the board, and require the pupils to distinguish them.

When I say, "I *crush* the worm," I express an action that is going on now, or in present time. "I *crushed* the worm," expresses an action that took place in past time. As *tense* means time, we call the form *crush* the present tense of the verb, and *crushed* the past tense. In the sentence, "The worm *crushed* under my foot died," *crushed*, expressing the action as assumed, is, as you have already learned, a participle; and, as the action is completed, we call it a past participle. Now notice that **ed** was added to *crush*, the verb in the present tense, to form the verb in the past tense, and to form the past participle. Most verbs form their past tense and their past participle by adding **ed**, and so we call such **Regular Verbs**.

"I *see* the man;" "I *saw* the man;" "The man *seen*

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

164

Graded Lessons In English.

by me ran away." "I *catch* fish in the brook ;" "I *caught* fish in the brook ;" "The fish *caught* in the brook tasted good." Here the verbs *see* and *catch* do not form their past tense and past participle by adding **ed** to the present, and hence we call them **Irregular Verbs**.

Let the teacher write on the board verbs of both classes, and require the pupils to distinguish them.

DEFINITIONS.

CLASSES OF VERBS WITH RESPECT TO MEANING.

A **Transitive Verb** is one that requires an object.¹

An **Intransitive Verb** is one that does not require an object.

CLASSES OF VERBS WITH RESPECT TO FORM.

A **Regular Verb** is one that forms its past tense and past participle by adding **ed** to the present.²

¹ The object of a transitive verb, that is, the name of the receiver of the action, may be the object complement, or it may be the subject; as, "Brutus stabbed *Cæsar*," "*Cæsar* was stabbed by Brutus."

² If the present ends in **e**, the **e** is dropped when **ed** is added; as love, loved; believe, believed.

It is quite common to classify verbs as **weak** and **strong** rather than as **regular** and **irregular**. Weak verbs are those that form their past tense by adding **ed**—or some form of it, as **d** or **t**—to the present; strong verbs are those that form their past tense by **vowel-change** alone. The full ending of the past participle weak is **ed**, and of the past participle strong is **en**.

Regular and **irregular**, if used, would denote those verbs that (1) **do**, and those that (2) **do not**, conform perfectly to the two types. *Fall, fell,*

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Composition.

165

An **Irregular Verb** is one that does not form its past tense and past participle by adding **ed** to the present.

COMPOSITION.

Place the following verbs in two columns, one headed transitive and the other intransitive. Place the same verbs in two other columns, one headed regular and the other irregular. Build these verbs into sentences by supplying a subject to each intransitive verb, and a subject and an object to each transitive verb:—

Vanish, gallop, bite, promote, contain, produce, provide, veto, secure, scramble, rattle, draw.

Arrange the following verbs as before, and then build them into sentences by supplying a subject and a noun attribute to each intransitive verb, and a subject and an object to each transitive verb:—

Degrade, gather, know, was, became, is.

A verb may be transitive in one sentence and intransitive in another. Use the following verbs both ways:—

Model. — The wren *sings* sweetly.

The wren *sings* a pretty little song.

Bend, ring, break, dash, move.

fallen would be a regular strong verb; and *walk*, *walked*, *walked*, a regular weak verb. *Wear*, *wore*, *worn* would be an irregular strong verb: and *creep*, *crept*, *crept*, an irregular weak verb.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

166

Graded Lessons In English.

LESSON 75.

CLASSES OF ADVERBS.

Hints for Oral Instruction. — When I say, "He will come *soon*, or *presently*, or *often*, or *early*," I am using, to modify *will come*, words which express the time of coming. These and all such adverbs we call **Adverbs of Time**.

"He will come *up*, or *hither*, or *here*, or *back*." Here I use, to modify *will come*, words which express place. These and all such adverbs we call **Adverbs of Place**.

When I say, "The weather is *so* cold, or *very* cold, or *intensely* cold," the words *so*, *very*, and *intensely* modify the adjective *cold* by expressing the degree of coldness. These and all such adverbs we call **Adverbs of Degree**.

When I say, "He spoke *freely*, *wisely*, and *well*," the words *freely*, *wisely*, and *well* tell how or in what manner he spoke. All such adverbs we call **Adverbs of Manner**.

Let the teacher place adverbs on the board, and require the pupil to classify them.

DEFINITIONS.

Adverbs of Time are those that generally answer the question, *When* ?

Adverbs of Place are those that generally answer the question, *Where* ?

Adverbs of Degree are those that generally answer the question, *To what extent* ?

Adverbs of Manner are those that generally answer the question, *In what way* ?

Digitized by Google

COMPOSITION.

Place the following adverbs in the four classes we have made—if the classification be perfect, there will be five words in each column—then build each adverb into a simple sentence:—

Partly, only, too, wisely, now, here, when, very, well, where, nobly, already, seldom, more, ably, away, always, not, there, out.

Some adverbs, as you have already learned, modify two verbs, and thus connect the two clauses in which these verbs occur. Such adverbs are called **Conjunctive Adverbs**.

The following dependent clauses are introduced by conjunctive adverbs. Build them into complex sentences by supplying independent clauses:—

— *when* the ice is smooth ; — *while* we sleep ; — *before* winter comes ; — *where* the reindeer lives ; — *wherever* you go.

LESSON 76.

CLASSES OF CONJUNCTIONS.

Hints for Oral Instruction. — “*Frogs, antelopes, and kangaroos can jump.*” Here the three nouns are of the same rank in the sentence. All are subjects of *can jump*. “*War has ceased, and peace has come.*” In this compound sentence, there are two clauses of the same rank. The word *and* connects the subjects of *can jump*, in the first sentence; and the two clauses, in the second. All words

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

168

Graded Lessons in English.

that connect words, phrases, or clauses of the same rank are called **Coördinate Conjunctions**.

"*If you have tears, prepare to shed them now;*" "*I will go, because you need me.*" Here *if* joins the clause, *you have tears*, as a modifier expressing condition, to the independent clause, "*prepare to shed them now*"; and *because* connects "*you need me*," as a modifier expressing reason or cause, to the independent clause, "*I will go*." These and all such conjunctions as connect dependent clauses to clauses of a higher rank are called **Subordinate Conjunctions**.

Let the teacher illustrate the meaning and use of the words *subordinate* and *coördinate*.

DEFINITIONS.

Coördinate Conjunctions are such as connect words, phrases, or clauses of the same rank.

Subordinate Conjunctions are such as connect clauses of different rank.

COMPOSITION.

Build four short sentences for each of the three coördinate conjunctions that follow. In the first, let the conjunction be used to connect principal parts of a sentence; in the second, to connect word modifiers; in the third, to connect phrase modifiers; and in the fourth, to connect independent clauses: —

And, or, but.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Classes of Connectives.

169

Write four short complex sentences containing the four subordinate conjunctions that follow. Let the first be used to introduce a noun clause, and the others to connect adverb clauses to independent clauses: —

That, for, if, because.

LIST OF CONNECTIVES.

Remark. — Some of the connectives below are conjunctions proper; some are relative pronouns; and some are adverbs or adverb phrases, which, in addition to their office as modifiers, may, in the absence of the conjunction, take its office upon themselves and connect the clauses.

COÖRDINATE CONNECTIVES.

Copulative. — *And, both . . . and, as well as*¹ are conjunctions proper. *Accordingly, also, besides, consequently, furthermore, hence, likewise, moreover, now, so, then, and therefore* are conjunctive adverbs.

Adversative. — *But and whereas* are conjunctions proper. *However, nevertheless, notwithstanding, on the contrary, on the other hand, still, and yet* are conjunctive adverbs.

Alternative. — *Neither, nor, or, either . . . or, and neither . . . nor* are conjunctions proper. *Else and otherwise* are conjunctive adverbs.

SUBORDINATE CONNECTIVES.

CONNECTIVES OF ADJECTIVE CLAUSES.

That, what, whatever, which, whichever, who, and whoever are relative pronouns. *When, where, whereby, wherein, and why* are conjunctive adverbs.

¹ The *as well as* in "He, *as well as* I, went"; and not that in "He is *as well as* I am."

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

170

Graded Lessons in English.

CONNECTIVES OF ADVERB CLAUSES.

Time. — *After, as, before, ere, since, till, until, when, whenever, while, and whilst* are conjunctive adverbs.

Place. — *Whence, where, and wherever* are conjunctive adverbs.

Degree. — *As, than, that, and the* are conjunctive adverbs, correlative with adjectives or adverbs.

Manner. — *As* is a conjunctive adverb, correlative often with an adjective or an adverb.

Real Cause. — *As, because, for, since, and whereas* are conjunctions proper.

Evidence. — *Because, for, and since* are conjunctions proper.

Purpose. — *In order that, lest (= that not), that, and so that* are conjunctions proper.

Condition. — *Except, if, in case that, on condition that, provided, provided that, and unless* are conjunctions proper.

Concession. — *Although, if (= even if), notwithstanding, though, and whether* are conjunctions proper. *However* is a conjunctive adverb. *Whatever, whichever, and whoever* are relative pronouns used indefinitely.

CONNECTIVES OF NOUN CLAUSES.

If, lest, that, and whether are conjunctions proper. *What, which, and who* are pronouns introducing questions; *how, when, whence, where, and why* are conjunctive adverbs.

LESSON 77.

REVIEW QUESTIONS.

What new subject begins with Lesson 74? Name and define the different classes of nouns. Illustrate by examples the difference between common nouns and proper nouns. Name and define the

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Composition.

171

different classes of pronouns. Can the pronoun *I* be used to stand for the one spoken to?—the one spoken of? Does the relative pronoun distinguish by its form the speaker, the one spoken to, and the one spoken of? Illustrate. What office is performed by a relative pronoun besides that of representing some antecedent noun or pronoun? Illustrate. Can any other class of pronouns be used to connect clauses?

For what do interrogative pronouns stand? Illustrate. Where may the antecedent of an interrogative pronoun generally be found? *Ans.*—The antecedent of an interrogative pronoun may generally be found in the answer to the question.

Name and define the different classes of adjectives. Give an example of each class. Name and define the different classes of verbs, made with respect to their meaning. Give an example of each class. Name and define the different classes of verbs, made with respect to their form. Give an example of each class.

Name and define the different classes of adverbs. Give examples of each kind. Name and define the different classes of conjunctions. Illustrate by examples.

Are prepositions and interjections subdivided? (See "Schemes" for the conjunction, the preposition, and the interjection, p. 270.)

COMPOSITION OF SENTENCES AND OF PARAGRAPHS.

ADAPTED FROM DR. JOHN BROWN—"RAB AND HIS FRIENDS."

Rab belonged to a lost tribe—there are no such dogs now. He was old and gray and brindled; and his hair short, hard, and close, like a lion's. He was as big as a Highland bull, and his body was thickset. He must have weighed ninety pounds at least.

His large, blunt head was scarred with the record of old wounds, a series of battlefields all over it. His muzzle was as black as night, his mouth blacker than any night, and a tooth or two, all he had,

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

172

Graded Lessons In English.

gleamed out of his jaws of darkness. One eye was out, one ear cropped close. The remaining eye had the power of two ; and above it, and in constant communication with it, was a tattered rag of an ear that was forever unfurling itself like an old flag.

And then that bud of a tail, about an inch long, if it could in any sense be said to be long, being as broad as it was long ! The mobility of it, its expressive twinklings and winkings, and the intercommunications between the eye, the ear, and it were of the oddest and swiftest.

Rab had the dignity and simplicity of great size. Having fought his way all along the road to absolute supremacy, he was as mighty in his own line as Julius Cæsar or the Duke of Wellington in his, and he had the gravity of all great fighters.

To the Teacher. — On the uses of words we suggest exercises similar to those preceding. Before attempting this it may be well to let the pupils go over these condensed expressions and supply the words necessary to the analysis. For instance, in the first paragraph *hair* may be followed by *was* and *Highland bull* by *is big*. In the next paragraph *wounds* may be followed by *marking*, as *night* by *is black*, etc. In the third paragraph *and then* may be followed by *there was*, etc. The pupils will determine whether supplying these words makes the description stronger or weaker.

Pupils may note especially the offices of nouns, verbs, and adjectives. This selection abounds in descriptive nouns and verbs that are particularly well chosen. Let the pupils point out such.

The Description. — How does the description above impress you ? Are only characteristic parts and features selected ? Are these few features enough to give you a distinct and vivid picture of Rab ? What comparisons do you find ? How do they help ? Pick out some words or phrases that seem to you very expressive. Find some words that are used, not in their first or common sense, but in a figurative sense. How do they help ?

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Review-Composition.

173

Paragraphs. — Which paragraph puts before you the dog as a whole? Where must this paragraph naturally stand? Why? Which paragraph describes Rab's character? What does each of the other paragraphs describe? If you think the arrangement of paragraphs above is the best, tell why.

Make a framework for this description.

ORIGINAL COMPOSITION.

Write a description of some animal which you have closely observed and in which you are interested. Be careful to pick out leading or characteristic features that will bring others into the reader's imagination. First prepare a framework.

REVIEW — COMPOSITION.

We recommend that the teacher select some short article containing valuable information and break up each paragraph into short, disconnected expressions. One paragraph at a time may be put on the board for the pupils to copy. The general subject may be given, and the pupils may be required to find a proper heading for the paragraph. The different ways of connecting the expressions may be discussed in the class. By contracting, expanding, and transposing, and by substituting entirely different words, a great variety of forms may be had. (The list of connectives in Lesson 76 may be helpful.) The pupils may then combine the different paragraphs into a composition.

We give, below, material for one composition, or theme: —

Frog's spawn found in a pond. At first like a mass of jelly. Eggs can be distinguished.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

174

Graded Lessons in English.

In a few days curious little fish are hatched. These "tadpoles" are lively. Swim by means of long tails. Head very large — out of proportion. Appearance of all head and tail. This creature is a true fish. It breathes water-air by means of gills. It has a two-chambered heart.

Watch it day by day. Two little gills seen. These soon disappear. Hind legs begin to grow. Tail gets smaller. Two small arms, or forelegs, are seen. Remarkable change going on inside. True lungs for breathing air have been forming. Another chamber added to the heart.

As the gills grow smaller, it finds difficulty in breathing water-air. One fine day it pokes its nose out of the water. Astonished (possibly) to find that it can breathe in the air. A new life has come upon it. No particular reason for spending all its time in water; crawls out upon land; sits down upon its haunches; surveys the world. It is no longer a fish; has entered upon a higher stage of existence; has become a frog.

This work of analyzing a composition to find the leading thoughts under which the other thoughts may be grouped is in many ways a most valuable discipline.

It teaches the pupil to compare, to discriminate, to weigh, to systematize, to read intelligently and profitably.

The reading-book will afford excellent practice in finding heads for paragraphs. Such work is an essential preparation for the reading class.

This composition work should serve as a constant review of all that has been passed over in the text-book.

Digitized by Google

MODIFICATIONS OF THE PARTS OF SPEECH

LESSON 78.

NOUNS AND PRONOUNS.

Hints for Oral Instruction. — You have learned that two words may express a thought, and that the thought may be varied by adding modifying words. You are now to learn that the meaning or use of a word may sometimes be changed by simply changing its form. The English language has lost many of its inflections, or forms, so that frequently changes in the meaning and use of words are not marked by changes in form. These changes in the form, meaning, and use of the parts of speech we call their **Modifications**.

“The *boy* shouts ;” “The *boys* shout.” I have changed the form of the subject *boy* by adding an *s* to it. The meaning has changed. *Boy* denotes one lad ; *boys*, two or more lads. This change in the form and meaning of nouns is called **Number**. The word *boy*, denoting one thing, is in the **Singular Number** ; and *boys*, denoting more than one thing, is in the **Plural Number**.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

176

Graded Lessons in English.

Let the teacher write other nouns on the board, and require the pupils to form the plural of them.

DEFINITIONS.

Modifications of the Parts of Speech are changes in their form, meaning, and use.

NUMBER.

Number is that modification of a noun or pronoun which denotes one thing or more than one.

The **Singular Number** denotes one thing.

The **Plural Number** denotes more than one thing.

RULE. — The **plural** of nouns is regularly formed by adding **s** or **es** to the singular.

Write the plural of the following nouns : —

Tree, bird, insect, cricket, grasshopper, wing, stick, stone, flower, meadow, pasture, grove, worm, bug, cow, eagle, hawk, wren, plow, shovel.

When a singular noun ends in the sound of **s**, **x**, **z**, **sh**, or **ch**, it is not easy to add the sound of **s**, so **es** is added to make another syllable.

Write the plural of the following nouns : —

Guess, box, topaz, lash, birch, compass, fox, waltz, sash, bench, gas, tax, adz, brush, arch.

Many nouns ending in **o** preceded by a consonant form the plural by adding **es** without increasing the number of syllables.

Digitized by Google

Nouns and Pronouns.—Number.

177

Write the plural of the following nouns : —

Hero, cargo, negro, potato, echo, volcano, mosquito, motto.

Common nouns ending in **y** preceded by a consonant form the plural by changing **y** into **i** and adding **es** without increasing the number of syllables.

Write the plural of the following nouns : —

Lady, balcony, family, city, country, daisy, fairy, cherry, study, sky.

Some nouns ending in **f** and **fe** form the plural by changing **f** or **fe** into **ves** without increasing the number of syllables.

Write the plural of the following nouns : —

Sheaf, loaf, beef, thief, calf, half, elf, shelf, self, wolf, life, knife, wife.

LESSON 79.

NUMBER.

From the following list of nouns, select and write in separate columns (1) those that have no plural; (2) those that have no singular; (3) those that are alike in both numbers : —

Pride, wages, trousers, cider, suds, victuals, milk, riches, flax, courage, sheep, deer, flour, idleness, tidings, thanks, axes, scissors, swine, heathen.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

178

Graded Lessons in English.

The following nouns have very irregular plurals—six changing the internal vowel, and two adding **en**. Learn to spell the plurals:—

<i>Singular.</i>	<i>Plural.</i>	<i>Singular.</i>	<i>Plural.</i>
Man,	men.	Foot,	feet.
Louse,	lice.	Ox,	oxen.
Child,	children.	Tooth,	teeth.
Mouse,	mice.	Goose,	geese.

Learn the following plurals and compare them with the groups in the preceding Lesson: —

Moneys, flies, chimneys, valleys, stories, berries, lilies, turkeys, monkeys, cuckoos, pianos, vetoes, solos, folios, gulfs, chiefs, leaves, roofs, scarfs, inches.

LESSON 80.

NOUNS AND PRONOUNS.—GENDER.

Hints for Oral Instruction. — “The *lion* was caged;” “The *lioness* was caged.” In the first sentence, something is said about a male lion; and in the second, something is said about a female lion. Modifications of the noun to denote the sex of the object we call **Gender**. Knowing the sex of the object, you know the gender of its English name. The word *lion*, denoting a male animal, is in the **Masculine Gender**; and *lioness*, denoting a female lion, is in the **Feminine Gender**.

The names of things without sex are in the **Neuter Gender**.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Such words as *cousin*, *child*, *friend*, *neighbor* may be either masculine or feminine.

DEFINITIONS.

Gender is that modification of a noun or pronoun which denotes sex.

The **Masculine Gender** denotes the male sex.

The **Feminine Gender** denotes the female sex.

The **Neuter Gender** denotes want of sex.

The feminine is distinguished from the masculine in three ways : —

- (1) By a difference in the ending of the nouns.
- (2) By different words in the compound names.
- (3) By words wholly or radically different.

Arrange the following pairs in separate columns with reference to these ways : —

Abbot, abbess ; actor, actress ; Francis, Frances ; Jesse, Jessie ; bachelor, maid ; beau, belle ; monk, nun ; gander, goose ; administrator, administratrix ; baron, baroness ; count, countess ; czar, czarina ; don, donna ; boy, girl ; drake, duck ; lord, lady ; nephew, niece ; landlord, landlady ; gentleman, gentlewoman ; peacock, peahen ; duke, duchess ; hero, heroine ; host, hostess ; Jew, Jewess ; man-servant, maid-servant ; sir, madam ; wizard, witch ; marquis, marchioness ; widower, widow ; heir, heiress ; Paul, Pauline ; Augustus, Augusta.

REVIEW QUESTIONS

What new way of varying the meaning of words is introduced in Lesson 78 ? Illustrate. What are modifications of the parts of speech ? What is number ? How many numbers are there ? Name and define

each. Give the rule for forming the plural of nouns. Illustrate the variations of this rule. What is gender? How many genders are there? Name and define each. In how many ways are the genders distinguished? Illustrate.

LESSON 81.

NOUNS AND PRONOUNS. — PERSON AND CASE.

Hints for Oral Instruction. — Number and gender, as you have already learned, are modifications affecting the meaning of nouns and pronouns. Number is almost always indicated by the ending; gender, sometimes. There are two other modifications which refer not to changes in the meaning of nouns and pronouns but to their different uses and relations. In the English language, these changes are not often indicated by a change of form.

"*I Paul* have written;" "*Paul, thou* art beside thyself;" "*He* brought *Paul* before Agrippa." In these three sentences the word *Paul* has three different uses. In the first, it is used as the name of the speaker; in the second, as the name of one spoken to; in the third, as the name of one spoken of. You will notice that the form of the noun is not changed. This change in the use of nouns and pronouns is called **Person**. The word *I* in the first sentence, the word *thou* in the second, and the word *he* in the third have each a different use. *I, thou, and he* are personal pronouns, and, as you have learned, distinguish person by their form. *I*, denoting the speaker, is

in the **First Person** ; *thou*, denoting the one spoken to, is in the **Second Person** ; and *he*, denoting the one spoken of, is in the **Third Person**.

Personal pronouns and verbs are the only words that distinguish person by their form.

"The *bear killed the man*;" "The *man killed the bear*;" "The *bear's grease* was made into hair oil." In the first sentence, the bear is represented as performing an action ; in the second, as receiving an action ; in the third, as possessing something. Hence the word *bear* in these sentences has three different uses. These uses of nouns are called **Cases**. The use of a noun as subject is called the **Nominative Case** ; its use as object is called the **Objective Case** ; and its use to denote possession is called the **Possessive Case**.

The possessive is the only case of nouns that is indicated by a change in form.

A noun or pronoun used as an attribute complement is in the nominative case. A noun or pronoun following a preposition as the principal word of a phrase is in the objective case. *I* and *he* are nominative forms. *Me* and *him* are objective forms.

The following sentences are therefore incorrect : It is *me* ; It is *him* ; *Me* gave the pen to *he*.

DEFINITIONS.

Person is that modification of a noun or pronoun which denotes the speaker, the one spoken to, or the one spoken of.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

182

Graded Lessons in English.

The **First Person** denotes the one speaking.

The **Second Person** denotes the one spoken to.

The **Third Person** denotes the one spoken of.

Case is that modification of a noun or pronoun which denotes its office in the sentence.

The **Nominative Case of a noun or pronoun** denotes its office as subject or as attribute complement.

The **Possessive Case of a noun or pronoun** denotes its office as possessive modifier.

The **Objective Case of a noun or pronoun** denotes its office as object complement, or as principal word in a prepositional phrase.

LESSON 82.

NOUNS AND PRONOUNS.—PERSON AND CASE.

Tell the person and case of each of the following nouns and pronouns — **remembering** that a noun or pronoun used as an explanatory modifier is in the same case as the word which it explains, and that a noun or pronoun used independently is in the nominative case : —

We Americans do things in a hurry.

You Englishmen take more time to think.

The Germans do their work with the most patience and deliberation.

We boys desire a holiday.

Come on, my men ; I will lead you.

I, your teacher, desire your success.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Nouns and Pronouns.—Person and Case.

183

You, my pupils, are attentive.

I called on Tom, the tinker.

Friends, countrymen, and lovers, hear me for my cause.

Write simple sentences in which each of the following nouns shall be used in the three persons and in the three cases :—

Andrew Jackson, Alexander, Yankees.

Write a sentence containing a noun in the nominative case, used as an attribute; one in the nominative, used as an explanatory modifier; one in the nominative, used independently.

Write a sentence containing a noun in the objective case, used to complete two predicate verbs; one used to complete a participle; one used to complete an infinitive; one used with a preposition to make a phrase: one used as an explanatory modifier.

If the class is sufficiently mature, the objective complement may here be treated. This explanation may be of service :—

In "It made him *sad*," *made* does not fully express the action performed upon him—not "*made him*," but "*made sad* (saddened) *him*." *Sad* helps *made* to express the action, and also denotes a quality which, as the result of the action, belongs to the person represented by the object *him*.

Whatever completes the predicate and belongs to the object we call an **Objective Complement**.

Digitized by Google

Nouns, infinitives, and participles may be used in the same way, as: —

They made *Victoria queen* ;
It made him *weep* ;
It kept him *laughing*.

They | made / queen : Victoria **Explanation.** — The line that separates *made* from *queen* slants toward the object complement to show that *queen* belongs to the object.

A noun or pronoun used as objective complement is in the objective case.

The teacher may here explain such constructions as “I proved it to be *him*,” in which *it* is object complement and *to be him* is objective complement. *Him*, the attribute complement of *be*, is in the objective case, because *it*, the assumed subject of *be*, is objective. Let the pupils compare, “I proved it to be *him*” with “I proved that it was *he*”; “*Whom* did you suppose it to be?” with “*Who* did you suppose it was?” etc.

These uses of nouns and pronouns may here be introduced, if the class be sufficiently mature: —

1. He gave *John* a book.
2. He bought *me* a book.

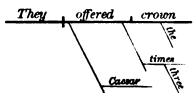
John and *me*, as here used, are called **Indirect Objects**. The indirect object names the one to or for whom something is done. We treat these words as modifiers without the preposition. If we change the order, the preposition must be supplied; as, “He gave a book *to John* ;” “He bought a book *for me*.”

Nouns denoting **measure, quantity, weight, time, value, distance, or direction** may be used **adverbially**, being equivalent to phrase modifiers without the preposition, as : —

1. We walked four *miles* an *hour*.
2. It weighs one *pound*.
3. It is worth a *dollar*.
4. The wall is ten *feet* six *inches* high.
5. I went *home* that *way*.

The following diagram will illustrate both the **indirect object** and the **noun of measure** : —

They offered Cæsar the crown three times.



Explanation. — *Cæsar* (the indirect object) and *times* (denoting measure) stand in the diagram on lines representing the principal words of prepositional phrases.

LESSON 83.

NOUNS AND PRONOUNS. — DECLENSION.

DEFINITION. — **Declension** is the arrangement of the cases of nouns and pronouns in the two numbers.

DECLENSION OF NOUNS.

LADY.		CHILD.	
<i>Singular.</i>	<i>Plural.</i>	<i>Singular.</i>	<i>Plural.</i>
<i>Nom.</i> lady,	ladies,	<i>Nom.</i> child,	children,
<i>Pos.</i> lady's,	ladies',	<i>Pos.</i> child's,	children's,
<i>Obj.</i> lady ;	ladies.	<i>Obj.</i> child ;	children.

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"

186

Graded Lessons in English.

DECLENSION OF PRONOUNS.

PERSONAL PRONOUNS.

FIRST PERSON.

<i>Singular.</i>	<i>Plural.</i>
<i>Nom.</i> I,	we,
<i>Pos.</i> my <i>or</i> mine,	our <i>or</i> ours,
<i>Obj.</i> me ;	us.

SECOND PERSON — common form.

<i>Singular.</i>	<i>Plural.</i>
<i>Nom.</i> you,	you,
<i>Pos.</i> your <i>or</i> yours,	your <i>or</i> yours,
<i>Obj.</i> you ;	you.

SECOND PERSON — old form.

<i>Singular.</i>	<i>Plural.</i>
<i>Nom.</i> thou,	ye <i>or</i> you,
<i>Pos.</i> thy <i>or</i> thine,	your <i>or</i> yours,
<i>Obj.</i> thee ;	you.

THIRD PERSON — masculine.

<i>Singular.</i>	<i>Plural.</i>
<i>Nom.</i> he,	they.
<i>Pos.</i> his,	their <i>or</i> theirs,
<i>Obj.</i> him ;	them.

THIRD PERSON — feminine.

<i>Singular.</i>	<i>Plural.</i>
<i>Nom.</i> she,	they,
<i>Pos.</i> her <i>or</i> hers,	their <i>or</i> theirs,
<i>Obj.</i> her ;	them.

Digitized by Google

Nouns and Pronouns.—Declension.

187

THIRD PERSON — neuter.

<i>Singular.</i>	<i>Plural.</i>
<i>Nom.</i> it,	they,
<i>Pos.</i> its,	their or theirs,
<i>Obj.</i> it ;	them.

Mine, ours, yours, thine, hers, and theirs are used when the name of the thing possessed is omitted ; as, "This rose is *yours*" = "This rose is *your rose*."

COMPOUND PERSONAL PRONOUNS.

By joining the word *self* to the possessive forms *my, thy, your* and to the objective forms *him, her, it*, the **Compound Personal Pronouns** are formed. They have no possessive case, and are alike in the nominative and the objective.

Their plurals are *ourselves, yourselves, and themselves*. Form the compound personal pronouns, and write their declension.

RELATIVE AND INTERROGATIVE PRONOUNS.

<i>Sing. and Plu.</i>	<i>Sing. and Plu.</i>
<i>Nom.</i> who,	<i>Nom.</i> which,
<i>Pos.</i> whose,	<i>Pos.</i> whose,
<i>Obj.</i> whom.	<i>Obj.</i> which.

Of which is often used instead of the possessive form of the latter pronoun. In actual use, *whose*, interrogative, is the possessive of *who* only.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

188

Graded Lessons in English.

<i>Sing. and Plu.</i>	<i>Sing. and Plu.</i>
<i>Nom.</i> that,	<i>Nom.</i> what,
<i>Pos.</i> —,	<i>Pos.</i> —,
<i>Obj.</i> that.	<i>Obj.</i> what.

Ever and *soever* are added to *who*, *which*, and *what* to form the **Compound Relative Pronouns**. They are used when the antecedent is omitted. For declension see above.

LESSON 84.

POSSESSIVE FORMS.

RULE.—The **possessive case** of nouns is formed in the singular by adding to the nominative the apostrophe and the letter *s* ('*s*'); in the plural, by adding the apostrophe ('') only. If the plural does not end in *s*, the apostrophe and the *s* are both added.

Write the possessive singular and the possessive plural of the following nouns, and place an appropriate noun after each:—

Robin, friend, fly, hero, woman, bee, mouse, cuckoo, fox, ox, man, thief, fairy, mosquito, wolf, shepherd, farmer, child, neighbor, cow.

Possession may be expressed also by the preposition *of* and the objective; as, the *mosquito's* bill = the bill *of* the *mosquito*.

The possessive sign ('*s*') is confined chiefly to the names of persons and animals.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Possessive Forms.

189

We do not say the *chair's* legs, but the legs *of* the *chair*.
Regard must be had also to the sound.

Improve the following expressions, and expand each into a simple sentence :—

The sky's color ; the cloud's brilliancy ; the rose's leaves ; my uncle's partner's house ; George's father's friend's farm ; the mane of the horse of my brother ; my brother's horse's mane.

When there are several possessive nouns, all belonging to one word, the possessive sign is added to the last only. If they modify different words, the sign is added to each.

Correct the following expressions, and expand each into a simple sentence :—

Model.—*Webster and Worcester's* dictionary may be bought at *Ticknor's and Field's* bookstore.

The possessive sign should be added to *Webster*, for the word *dictionary* is understood immediately after. *Webster and Worcester* did not together possess the same dictionary. The sign should not be added to *Ticknor*, for the two men, *Ticknor and Field*, owned the same store.

Adam's and Eve's garden ; Jacob's and Esau's father ; Shakespeare and Milton's works ; Maud, Kate, and Clara's gloves ; Maud's. Kate's, and Clara's teacher was ———.

When one possessive noun is explanatory of another, the possessive sign is added to the last only.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

190

Graded Lessons in English.

Correct the following errors : —

I called at Tom's the tinker's. They listened to Peter's the Hermit's eloquence. This was the Apostle's Paul's advice.

Correct the following errors : —

Our's, your's, hi's, their's, her's, it's, hien, youm, hern.

LESSON 85.

FORMS OF THE PRONOUN.

Remember that *I, we, thou, ye, he, she, they*, and *who* are **nominative** forms, and must not be used in the objective case.

Remember that *me, us, thee, him, her, them*, and *whom* are **objective** forms, and must not be used in the nominative case.

To the Teacher. — The eight nominative forms and the seven objective forms given above are the only distinctive nominative and objective forms in the English language. Let the pupils become familiar with them

Correct the following errors : —

Him and me are good friends.

The two persons were her and me.

Us girls had a jolly time.

It is them, surely.

Who will catch this? Me.

Them that despise me shall be lightly esteemed

Who is there? Me.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Forms of the Pronoun.

191

It was not us, it was him.

Who did you see?

Who did you ask for?

Remember that pronouns must agree with their antecedents in number, gender, and person.

Correct the following errors :—

Every boy must read their own sentences.

I gave the horse oats, but he would not eat it.

Every one must read it for themselves.

I took up the little boy, and set it on my knee.

Remember that the relative *who* represents persons ; *which*, animals and things ; *that*, persons, animals, and things ; and *what*, things.

Correct the following errors :—

I have a dog who runs to meet me.

The boy which I met was quite lame.

Those which live in glass houses must not throw stones.

REVIEW QUESTIONS.

To the Teacher.— For "Schemes," see p. 268.

How many modifications have nouns and pronouns? Name and define each. How many persons are there? Define each. How many cases are there? Define each. How do you determine the case of an explanatory noun or pronoun? What is declension? How are the forms *mine*, *yours*, etc., now used? What is the rule for forming the possessive case? What words are used only in the nominative case? What words are used only in the objective case?¹ How do you determine the number, gender, and person of pronouns?

¹ *Her* is used in the possessive case also.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

192

Graded Lessons in English.

LESSON 86.

NOUNS AND PRONOUNS — PARSING.

To the Teacher. — For general "Scheme" for parsing, see p. 271.

Select and parse all the nouns and pronouns in Lesson 53.

Model for Written Parsing. — *Elizabeth's favorite, Raleigh, was beheaded by James I.*

CLASSIFICATION.		MODIFICATIONS.				SYNTAX.
<i>Nouns.</i>	<i>Kind.</i>	<i>Person.</i>	<i>Number.</i>	<i>Gender.</i>	<i>Case.</i>	
Elizabeth's	Prop.	3rd.	Sing.	Fem.	Pos.	Pos. Mod. of <i>favorite</i> .
favorite	Com.	"	"	Mas.	Nom.	Sub. of <i>was beheaded</i> .
Raleigh	Prop.	"	"	"	"	Exp. Mod. of <i>favorite</i> .
James I.	"	"	"	"	Obj.	Prin. word after <i>by</i> .

To the Teacher. — Select other exercises, and continue this work as long as it may be profitable. See Lessons 56, 57, 61, 64, and 65.

LESSON 87.

COMPARISON OF ADJECTIVES.

Adjectives have one modification ; viz., **Comparison**.

DEFINITIONS.

Comparison is a modification of the adjective to express the relative degree of the quality in the things compared.

The **Positive Degree** express the simple quality.

The **Comparative Degree** expresses a greater or a less degree of the quality.

Digitized by Google

The **Superlative Degree** expresses the greatest or the least degree of the quality.

RULE.— **Adjectives** are regularly compared by adding **er** to the positive to form the comparative, and **est** to the positive to form the superlative.

Adjectives of one syllable are generally compared regularly ; adjectives of two or more syllables are often compared by prefixing *more* and *most*. To express diminution, we prefix *less* and *least*.

When there are two correct forms, choose the one that can be more easily pronounced.

Compare the following adjectives. For the spelling consult your dictionaries : —

<i>Positive.</i>	<i>Comparative.</i>	<i>Superlative.</i>
Lovely,	lovelier,	loveliest ; or
lovely,	more lovely,	most lovely.

Tame, warm, beautiful, brilliant, amiable, high, mad, greedy, pretty, hot.

Some adjectives are compared irregularly. Learn the following forms : —

<i>Positive.</i>	<i>Comparative.</i>	<i>Superlative.</i>
Good,	better,	best.
Bad,		
Evil,	worse,	worst.
Ill,		
Little,	less,	least.
Much,	more,	most.
Many,		

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

194

Graded Lessons In English.

LESSON 88.

COMPARISON OF ADJECTIVES AND ADVERBS.

Remember that, when two things or groups of things are compared, the comparative degree is commonly used ; when more than two, the superlative is employed.

Caution. — Adjectives should not be doubly compared.

Correct the following errors : —

Of all the boys, George is the more industrious.

Peter ~~was~~ older than the twelve apostles.

Which is the longer of the rivers of America ?

This ~~was~~ the most unkindest cut of all.

He chose a more humbler part.

My hat is more handsomer than yours.

The younger of those three boys is the smarter.

Which is the more northerly, Maine, Oregon, or Minnesota ?

Caution. — Do not use adjectives and adverbs extravagantly.

Correct the following errors : —

The weather is horrid.

That dress is perfectly awful.

Your coat sits frightfully.

We had an awfully good time.

This is a tremendously hard lesson.

Harry is a mighty nice boy.

Remember that adjectives whose meaning does not admit of different degrees cannot be compared ; as, *every*, *infinite*.

Digitized by Google

Comparison of Adjectives and Adverbs. 195

Use in the three different degrees such of the following adjectives as admit of comparison : —

All, serene, excellent, immortal, first, two, total, universal, three-legged, bright. .

Adverbs are compared in the same manner as adjectives. The following are compared regularly. Compare them : —

Fast, often, soon, late, early.

In the preceding and in the following list, find words that may be used as adjectives.

The following are compared irregularly ; learn them : —

<i>Positive.</i>	<i>Comparative.</i>	<i>Superlative.</i>
Badly, } Ill, }	worse,	worst.
Well,	better,	best.
Little,	less,	least.
Much,	more,	most.
Far, ✓	farther,	farthest.

Adverbs ending in *ly* are generally compared by prefixing *more* and *most*. Compare the following : —

Firmly, gracefully, actively, easily. .

To the Teacher. — Let the pupils select and parse all the adjectives and adverbs in Lesson 27. Select other exercises, and continue the work as long as it is profitable. See "Schemes" for review, p. 270.

REVIEW QUESTIONS.

How is a noun parsed? What modifications have adjectives? What is comparison? How many degrees of comparison are there?

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

196

Graded Lessons in English.

Define each. How are adjectives regularly compared? Distinguish the uses of the comparative and the superlative degree. Give the directions for using adjectives and adverbs (Lesson 88). Illustrate. What adjectives cannot be compared? How are adverbs compared?

LESSON 89.

MODIFICATION OF VERBS.

VOICE.

Hints for Oral Instruction. — "*I picked the rose.*" The same thing may be said in another way. "*The rose was picked by me.*" The first verb, *picked*, shows that the subject represents the actor, and the second form of the verb, *was picked*, shows that the subject names the thing acted upon. This change in the form and use of the verb is called **Voice**. The first form is called the **Active Voice**; and the second, the **Passive Voice**.

The passive form is very convenient when we wish to assert an action without naming any actor. "*Money is coined*" is better than "*Somebody coins money.*"

DEFINITIONS.

Voice is that modification of the transitive verb which shows whether the subject names the **actor** or the **thing acted upon**.

The **Active Voice** shows that the subject names the actor.

The **Passive Voice** shows that the subject names the thing acted upon.

Digitized by Google

In each of the following sentences, change the voice of the verb without changing the meaning of the sentence, and note the other changes that occur in the sentence : —

The industrious bees gather honey from the flowers.

The storm drove the vessel against the rock.

Our words should be carefully chosen.

Death separates the dearest friends.

His vices have weakened his mind and destroyed his health.

True valor protects the feeble and humbles the oppressor.

The Duke of Wellington, who commanded the English armies in the Peninsula, never lost a battle.

Moses led the Israelites out of Egypt.

Dr. Livingstone explored a large part of Africa.

The English were conquered by the Normans.

Name all the transitive verbs in Lessons 20 and 22, and give their voice.

LESSON 90.

MODE, TENSE, NUMBER, AND PERSON.

Hints for Oral Instruction. — When I say, "James *walks*," I assert the walking as a fact. When I say, "James *may walk*," I do not assert the action as a fact, but as a possible action. When I say, "If James *walk* out, he will improve," I assert the action, not as an actual fact, but as a condition of James's improving — a condition that may or may not become a fact. When I say to James, "*Walk* out," I do not assert that James actually does the act, I assert the action as a command.

The action expressed by the verb *walk* is here asserted in four different ways, or **Modes**.¹ The first way is called the **Indicative Mode**; the second, the **Potential Mode**; the third, the **Subjunctive Mode**; the fourth, the **Imperative Mode**.

Let the teacher give other examples and require the pupils to repeat this instruction.

For the two forms of the verb called the **Infinitive** and the **Participle**, see "Hints," Lessons 48 and 49.

"*I walk*;" "*I walked*;" "*I shall walk*." In each of these sentences, the manner of asserting the action is the same. "*I walk*" expresses the action as present; "*I walked*" expresses the action as past; "*I shall walk*" expresses the action as future. As **Tense** means time, the first form is called the **Present Tense**; the second, the **Past Tense**; and the third, the **Future Tense**.

We have three other forms of the verb, expressing the action as completed in the present, the past, or the future.

"*I have walked* out to-day;" "*I had walked* out when he called;" "*I shall have walked* out by to-morrow." The form, *have walked*, expressing the action as completed in the present, is called the **Present Perfect Tense**. The form, *had walked*, expressing the action as completed in the past, is called the **Past Perfect Tense**. The form,

¹ Many grammarians reject the potential mode, insisting that, when we assert the power, liberty, or possibility of acting or being, we assert it (1) as a fact, and the verb is in the indicative; or we assert it (2) as a supposition or conception merely, and the verb is in the subjunctive.

shall have walked, expressing an action to be completed in the future, is called the **Future Perfect Tense**.

Let the teacher give other verbs, and require the pupils to name and explain the different tenses.

"*I walk*;" "*Thou walkest*;" "*He walks*;" "*They walk*." In the second sentence, the verb *walk* was changed by adding **est**; and in the third, it was changed by adding **s**. These changes are for the sake of agreement with the person of the subject. The verb ending in **est** agrees with the subject *thou* in the second person, and the verb ending in **s** agrees with *he* in the third person. In the fourth sentence, the subject is in the third person; but it is plural, and so the verb drops the **s** to agree with the plural *they*.

Verbs are said to agree in **Person** and **Number** with their subjects. The person and number forms may be found in Lessons 93, 94.

DEFINITIONS.

Mode is that modification of the verb which denotes the manner of asserting the action or being.

The **Indicative Mode** asserts the action or being as a fact.

The **Potential Mode** asserts the power, liberty, possibility, or necessity of acting or being.

The **Subjunctive Mode** asserts the action or being as a mere condition, supposition, or wish.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

200

Graded Lessons in English.

The **Imperative Mode** asserts the action or being as a command or an entreaty.

The **Infinitive** is a form of the verb which names the action or being in a general way, without asserting it of anything.

The **Participle** is a form of the verb partaking of the nature of an adjective or of a noun,¹ and expressing the action or being as assumed.

The **Present Participle** denotes action or being as continuing at the time indicated by the predicate.

The **Past Participle** denotes action or being as past or completed at the time indicated by the predicate.

The **Past Perfect Participle** denotes action or being as completed at a time previous to that indicated by the predicate.

Tense is that modification of the verb which expresses the time of the action or being.

The **Present Tense** expresses action or being as present.

The **Past Tense** expresses action or being as past.

The **Future Tense** expresses action or being as yet to come.

The **Present Perfect Tense** expresses action or being as completed at the present time.

The **Past Perfect Tense** expresses action or being as completed at some past time.

The **Future Perfect Tense** expresses action or being to be completed at some future time.

Number and **Person** of a verb are those modifications that show its agreement with the number and person of its subject.

¹See Lesson 98, foot-note.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Conjugation of the Verb.

201

LESSON 91.

CONJUGATION OF THE VERB.

DEFINITIONS.

Conjugation is the regular arrangement of all the forms of the verb.

Synopsis is the regular arrangement of the forms of one number and person in all the modes and tenses.

Auxiliary Verbs are those that help in the conjugation of other verbs.

The auxiliaries are *do, be, have, shall, will, may, can,* and *must*.

The **Principal Parts** of a verb are the present indicative or the present infinitive, the past indicative, and the past participle.

These are called principal parts, because all the other forms of the verb are derived from them.

We give, below, the principal parts of some of the most important irregular verbs.¹ Learn them.

<i>Present.</i>	<i>Past.</i>	<i>Past Par.</i>
Be or am,	was,	been.
Begin,	began,	begun.
Blow,	blew,	blown.
Break,	broke,	broken.
Choose,	chose,	chosen.
Come,	came,	come.
Do,	did,	done.
Draw,	drew,	drawn.

¹ Most of those in the list are irregular strong verbs. See Lesson 74, foot-note.

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"

202

Graded Lessons in English.

<i>Present.</i>	<i>Past.</i>	<i>Past Par.</i>
Drink,	drank,	drunk.
Drive,	drove,	driven.
Eat,	ate,	eaten.
Fall,	fell,	fallen.
Fly,	flew,	flown.
Freeze,	froze,	frozen.
Go,	went,	gone.
Get,	got,	got or gotten.
Give,	gave,	given.
Grow,	grew,	grown.
Know,	knew,	known.
Lay,	laid,	laid.
Lie (to rest),	lay,	lain.
Ride,	rode,	ridden.
Ring,	rang or rung,	rung.
Rise,	rose,	risen.
Run,	ran,	run.
See,	saw,	seen.
Set,	set,	set.
Sit,	sat,	sat.
Shake,	shook,	shaken.
Sing,	sang or sung,	sung.
Slay,	slew,	slain.
Speak,	spoke,	spoken.
Steal,	stole,	stolen.
Swim,	swam or swum,	swum.
Take,	took,	taken.
Tear,	tore,	torn.
Throw,	threw,	thrown.
Wear,	wore,	worn.
Write,	wrote,	written.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Conjugation of the Verb **SEE**.

203

The following irregular verbs are called **Defective** because some of their parts are wanting:—

<i>Present.</i>	<i>Past.</i>	<i>Present.</i>	<i>Past.</i>
Can,	could.	Will,	would.
May,	might.	Must,	—
Shall,	should.	Ought,	—

LESSON 92.

CONJUGATION OF THE VERB **SEE** IN THE SIMPLE FORM.

PRINCIPAL PARTS.

<i>Present.</i>	<i>Past.</i>	<i>Past Par.</i>
See,	saw,	seen.

Indicative Mode.

PRESENT TENSE.

<i>Singular.</i>	<i>Plural.</i>
1. I see,	1. We see,
2. { You see, <i>or</i> Thou seest,	2. You see,
3. He sees ;	3. They see.

PAST TENSE.

1. I saw,	1. We saw,
2. { You saw, <i>or</i> Thou sawest,	2. You saw,
3. He saw ;	3. They saw.

FUTURE TENSE.

1. I shall see,	1. We shall see,
2. { You will see, <i>or</i> Thou wilt see,	2. You will see,
3. He will see ;	3. They will see.

Digitized by Google

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"

204

Graded Lessons in English.

PRESENT PERFECT TENSE.

Singular.

1. I have seen,
2. { You have seen, *or*
Thou hast seen,
3. He has seen ;

Plural.

1. We have seen,
2. You have seen,
3. They have seen.

PAST PERFECT TENSE.

1. I had seen,
2. { You had seen, *or*
Thou hadst seen,
3. He had seen ;

1. We had seen,
2. You had seen,
3. They had seen.

FUTURE PERFECT TENSE.

1. I shall have seen,
2. { You will have seen, *or*
Thou wilt have seen,
3. He will have seen ;

1. We shall have seen,
2. You will have seen,
3. They will have seen.

Potential Mode.¹

PRESENT TENSE.

Singular.

1. I may see,
2. { You may see, *or*
Thou mayst see,
3. He may see ;

Plural.

1. We may see,
2. You may see,
3. They may see.

PAST TENSE.

1. I might see,
2. { You might see, *or*
Thou mightst see,
3. He might see ;

1. We might see,
2. You might see,
3. They might see.

¹ See Lesson 90, foot-note.

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"

Conjugation of the Verb SEE.

205

PRESENT PERFECT TENSE.

Singular.

Plural.

- | | |
|----------------------------|------------------------|
| 1. I may have seen, | 1. We may have seen, |
| 2. { You may have seen, or | 2. You may have seen, |
| { Thou mayst have seen, | |
| 3. He may have seen ; | 3. They may have seen. |

PAST PERFECT TENSE.

- | | |
|------------------------------|--------------------------|
| 1. I might have seen, | 1. We might have seen, |
| 2. { You might have seen, or | 2. You might have seen, |
| { Thou mightst have seen, | |
| 3. He might have seen ; | 3. They might have seen. |

Subjunctive Mode.

PRESENT TENSE.

Singular.

Plural.

- | | |
|---------------------|-----------------|
| 1. If I see, | 1. If we see, |
| 2. { If you see, or | 2. If you see, |
| { If thou see, | |
| 3. If he see ; | 3. If they see. |

Imperative Mode.

PRESENT TENSE.

- | | |
|------------------------|---------------|
| 2. See (you or thou) ; | 2. See (you). |
|------------------------|---------------|

Infinitives.

PRESENT TENSE.

To see.

PRESENT PERFECT TENSE.

To have seen.

Participles.

PRESENT.

PAST.

PAST PERFECT

Seeing.

Seen.

Having seen.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

206

Graded Lessons in English.

To the Teacher.—Let the pupils prefix *do* and *did* to the simple present *see*, and thus make the **Emphatic form** of the present and the past tense.

Let *can* and *must* be used in place of *may*; and *could*, *would*, and *should*, in place of *might*.

Require the pupils to tell how each tense is formed, and to note all changes for agreement in number and person.

A majority of modern writers use the indicative forms instead of the subjunctive in all of the tenses, unless it may be the present. The subjunctive forms of the verb *be* are retained in the present and the past tense.

Let the pupils understand that the mode and tense forms do not always correspond with the actual meaning. "The ship *sails* next week." "I *may go* to-morrow." The verbs *sails* and *may go* are present in form but future in meaning. "If it *rains* by noon, he may not come." The verb *rains* is indicative in form but subjunctive in meaning.

The plural forms, *You saw*, *You were*, etc., are used in the singular also.

LESSON 93.

CONJUGATION OF THE VERB.—SIMPLE FORM.

Fill out the following forms, using the principal parts of the verb *walk*. *Pres.*, *walk*; *Past*, *walked*; *Past Par.*, *walked*.

Indicative Mode.

PRESENT TENSE.

<i>Singular.</i>		<i>Plural.</i>	
1. I	<u>Pres.</u>	1. We	<u>Pres.</u>
2. { You	<u>Pres.</u>	2. You	<u>Pres.</u>
Thou	<u>Pres.</u> <i>est</i> ,	3. They	<u>Pres.</u>
3. He	<u>Pres.</u> <i>s</i> ;		

Digitized by Google

Conjugation of the Verb.—Simple Form.

207

PAST TENSE.

Singular.

Plural.

- | | |
|------------------------------|-----------------------|
| 1. I <u>Past</u> , | 1. We <u>Past</u> , |
| 2. { You <u>Past</u> , | 2. You <u>Past</u> , |
| Thou <u>Past</u> <i>st</i> , | 3. They <u>Past</u> . |
| 3. He <u>Past</u> ; | |

FUTURE TENSE.

- | | |
|-------------------------------------|------------------------------------|
| 1. I <i>shall</i> <u>Pres.</u> , | 1. We <i>shall</i> <u>Pres.</u> , |
| 2. { You <i>will</i> <u>Pres.</u> , | 2. You <i>will</i> <u>Pres.</u> , |
| Thou <i>will-t</i> <u>Pres.</u> , | 3. They <i>will</i> <u>Pres.</u> . |
| 3. He <i>will</i> <u>Pres.</u> ; | |

PRESENT PERFECT TENSE.

- | | |
|---|--|
| 1. I <i>have</i> <u>Past Par.</u> , | 1. We <i>have</i> <u>Past Par.</u> , |
| 2. { You <i>have</i> <u>Past Par.</u> , | 2. You <i>have</i> <u>Past Par.</u> , |
| Thou <i>ha-st</i> <u>Past Par.</u> , | 3. They <i>have</i> <u>Past Par.</u> . |
| 3. He <i>ha-s</i> <u>Past Par.</u> ; | |

PAST PERFECT TENSE.

- | | |
|--|---------------------------------------|
| 1. I <i>had</i> <u>Past Par.</u> , | 1. We <i>had</i> <u>Past Par.</u> , |
| 2. { You <i>had</i> <u>Past Par.</u> , | 2. You <i>had</i> <u>Past Par.</u> , |
| Thou <i>had-st</i> <u>Past Par.</u> , | 3. They <i>had</i> <u>Past Par.</u> . |
| 3. He <i>had</i> <u>Past Par.</u> ; | |

FUTURE PERFECT TENSE.

- | | |
|--|---|
| 1. I <i>shall have</i> <u>Past Par.</u> , | 1. We <i>shall have</i> <u>Past Par.</u> , |
| 2. { You <i>will have</i> <u>Past Par.</u> , | 2. You <i>will have</i> <u>Past Par.</u> , |
| Thou <i>will-t have</i> <u>Past Par.</u> , | 3. They <i>will have</i> <u>Past Par.</u> . |
| 3. He <i>will have</i> <u>Past Par.</u> ; | |

Digitized by Google

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"

208

Graded Lessons in English.

Potential Mode.

PRESENT TENSE.

Singular.

1. I *may* Pres.,
2. { You *may* Pres.,
Thou *may-st* Pres.,
3. He *may* Pres. ;

Plural.

1. We *may* Pres.,
2. You *may* Pres.,
3. They *may* Pres..

PAST TENSE.

1. I *might* Pres.,
2. { You *might* Pres.,
Thou *might-st* Pres.,
3. He *might* Pres. ;

1. We *might* Pres.,
2. You *might* Pres.,
3. They *might* Pres..

PRESENT PERFECT TENSE.

1. I *may have* Past Par.,
2. { You *may have* Past Par.,
Thou *may-st have* Past Par.,
3. He *may have* Past Par. ;

1. We *may have* Past Par.,
2. You *may have* Past Par.,
3. They *may have* Past Par..

PAST PERFECT TENSE.

1. I *might have* Past Par.,
2. { You *might have* Past Par.,
Thou *might-st have* Past Par.,
3. He *might have* Past Par. ;

1. We *might have* Past Par.,
2. You *might have* Past Par.,
3. They *might have* Past Par..

Subjunctive Mode.

PRESENT TENSE.

Singular.

1. If I Pres.,
2. { If you Pres.,
If thou Pres.,
3. If he Pres. ;

Plural.

1. If we Pres.,
2. If you Pres.,
3. If they Pres..

Digitized by Google

Conjugation of the Verb BE.

209

Imperative Mode.

PRESENT TENSE.

Singular.

2. Pres. (you or thou);

Plural.

2. Pres. (you).

Infinitives.

PRESENT TENSE.

To Pres..

PRESENT PERFECT TENSE.

To have Past Par..

Participles.

PRESENT.

Pres. *ing.*

PAST.

Past Par.

PAST PERFECT.

Having Past Par.

To the Teacher. — Let the pupils fill out these forms with other verbs. In the indicative, present, third, singular, *es* is sometimes added instead of *s*; and in the second person, old style, *st* is sometimes added instead of *est*.

LESSON 94.

CONJUGATION OF THE VERB BE.

In studying this Lesson, pay no attention to the line at the right of each verb.

Indicative Mode.

PRESENT TENSE.

Singular.

1. I am —,
2. { You are —, *or*
Thou art —,
3. He is —;

Plural.

1. We are —,
2. You are —,
3. They are —.

Digitized by Google

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"

210

Graded Lessons in English.

PAST TENSE.

Singular.

1. I was —,
2. { You were —, *or*
Thou wast —,
3. He was — ;

Plural.

1. We were —,
2. You were —,
3. They were —.

FUTURE TENSE.

1. I shall be —,
2. { You will be —, *or*
Thou wilt be —,
3. He will be — ;

1. We shall be —,
2. You will be —,
3. They will be —.

PRESENT PERFECT TENSE.

1. I have been —,
2. { You have been —, *or*
Thou hast been —,
3. He has been — ;

1. We have been —,
2. You have been —,
3. They have been —.

PAST PERFECT TENSE.

1. I had been —,
2. { You had been —, *or*
Thou hadst been —,
3. He had been — ;

1. We had been —,
2. You had been —,
3. They had been —.

FUTURE PERFECT TENSE.

1. I shall have been —,
2. { You will have been —, *or*
Thou wilt have been —,
3. He will have been — ;

1. We shall have been —,
2. You will have been —,
3. They will have been —.

Digitized by Google

Conjugation of the Verb BE.

211

Potential Mode.

PRESENT TENSE.

- | <i>Singular.</i> | <i>Plural.</i> |
|--|-------------------|
| 1. I may be —, | 1. We may be —, |
| 2. { You may be —, <i>or</i>
Thou mayst be —, | 2. You may be —, |
| 3. He may be —; | 3. They may be —. |

PAST TENSE.

- | | |
|--|---------------------|
| 1. I might be —, | 1. We might be —, |
| 2. { You might be —, <i>or</i>
Thou mightst be —, | 2. You might be —, |
| 3. He might be —, | 3. They might be —. |

PRESENT PERFECT TENSE.

- | | |
|--|--------------------------|
| 1. I may have been —, | 1. We may have been —, |
| 2. { You may have been —, <i>or</i>
Thou mayst have been —, | 2. You may have been —, |
| 3. He may have been —; | 3. They may have been —. |

PAST PERFECT TENSE.

- | | |
|--|----------------------------|
| 1. I might have been —, | 1. We might have been —, |
| 2. { You might have been —, <i>or</i>
Thou mightst have been —, | 2. You might have been —, |
| 3. He might have been —; | 3. They might have been —. |

Subjunctive Mode.

PRESENT TENSE.

- | <i>Singular.</i> | <i>Plural.</i> |
|--|------------------|
| 1. If I be —, | 1. If we be —, |
| 2. { If you be —, <i>or</i>
If thou be —, | 2. If you be —, |
| 3. If he be —; | 3. If they be —. |

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

212

Graded Lessons in English.

PAST TENSE.

Singular.

1. If I were —,
2. { If you were —, or
If thou wert —,
3. If he were —;

Plural.

1. If we were —,
2. If you were —,
3. If they were —.

Imperative Mode.

PRESENT TENSE.

2. Be (you or thou) —;
2. Be (you) —.

Infinitives.

PRESENT TENSE.

To be —.

PRESENT PERFECT TENSE.

To have been —.

Participles.

PRESENT.

Being —.

PAST.

Been.

PAST PERFECT.

Having been —.

To the Teacher. — After the pupils have become thoroughly familiar with the verb *be* as a principal verb, teach them to use it as an auxiliary in making the **Progressive Form** and the **Passive Form**.

The progressive form may be made by filling all the blanks with the present participle of some verb.

The passive form may be made by filling all the blanks with the past participle of a transitive verb.

Notice that there is no blank after the past participle.

In the progressive form, this participle is wanting; and in the passive form, it is the same as in the simple.

Digitized by Google

LESSON 95.

AGREEMENT OF THE VERB.

Remember that the verb must agree with its subject in number and person.

Give the person and number of each of the following verbs, and write sentences in which each form shall be used correctly : —

Common forms. — Does, has = ha(ve)s, is, am, are, was, were.

Old forms. — Seest, sawest, hast = ha(ve)st, wilt, mayst, mightst, art, wast.

When a verb has two or more subjects connected by *and*, it must agree with them in the plural. A similar rule applies to the agreement of the pronoun.

Correct the following errors : —

Poverty and obscurity *oppresses* him who thinks that *it is* oppressive.

Wrong ; the verb *oppresses* should be *oppress* to agree with its two subjects connected by *and*. The pronoun *it* should be *they* to agree with its two antecedents, and the verb *is* should be *are* to agree with *they*.

Industry, energy, and good sense is essential to success.

Time and tide waits for no man.

The tall sunflower and the little violet is turning its face to the sun.

The mule and the horse was harnessed together.

Every green leaf and every blade of grass seem grateful.

Wrong ; the verb *seem* should be singular ; for, when several singular subjects are preceded by *each*, *every*, or *no*, they are taken separately.

Correct the following errors : —

Each day and each hour bring their portion of duty.

Every book and every paper were found in their place.

When a verb has two or more singular subjects connected by *or* or *nor*, it must agree with them in the singular. A similar rule applies to the agreement of the pronoun.

Correct the following errors : —

One or the other have made a mistake in their statement.

Neither the aster nor the dahlia are cultivated for their fragrance.

Either the president or his secretary were responsible.

Neither Ann, Jane, nor Sarah are at home.

To foretell, or to express future time simply, the auxiliary *shall* is used in the first person, and *will* in the second and third ; but, when a speaker determines or promises, he uses *will* in the first person and *shall* in the second and third.

Correct the following errors : —

I will freeze if I do not move about.

You shall feel better soon, I think.

She shall be fifteen years old to-morrow.

I shall find it for you if you shall bring the book to me.

You will have it if I can get it for you.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Errors in the Form of the Verb.

215

He will have it if he shall take the trouble to ask for it.
He will not do it if I can prevent him.
I will drown, nobody shall help me.
I will be obliged to you if you shall attend to it.
We will have gone by to-morrow morning.
You shall disappoint your father if you do not return.
I do not think I will like the change.
Next Tuesday shall be your birthday.
You shall be late if you do not hurry.

LESSON 96.

ERRORS IN THE FORM OF THE VERB.

When the past tense and the past participle differ in form, they are often confounded in use ; as, "I *done* it ;" "I *seen* it."

If the pupils are required to construct short sentences, using the Past forms in Lesson 91 as predicates, and the Past Participle forms as modifiers or as completing words in compound verbs, they may reach some such conclusions as these : —

The Past is always an asserting, or predicate, word ; the Past Participle never asserts, but is used as an adjective modifier or as the completing word of a compound verb ; the Present may be used as a predicate or as an infinitive.

Copy, and repeat aloud, these exercises : —

1. *Lay* down your pen
2. *Lie* down, Rover.
3. I *laid* down my pen.
4. The dog then *lay* down.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

216

Graded Lessons in English.

- | | |
|--|------------------------------------|
| 5. I have <i>laid</i> down my pen. | 10. I <i>sat</i> down and rested. |
| 6. The dog has <i>lain</i> down. | 11. I have <i>set</i> it down. |
| 7. <i>Set</i> the pail down. | 12. I have <i>sat</i> down. |
| 8. <i>Sit</i> down and rest. | 13. My work was <i>laid</i> aside. |
| 9. I then <i>set</i> it down. | 14. I was <i>lying</i> down. |
| 15. The trap was <i>set</i> by the river. | |
| 16. I was <i>sitting</i> by the river. | |
| 17. The garment <i>sits</i> well. | |
| 18. The hen <i>sits</i> on her eggs. | |
| 19. He came in and <i>lay</i> down. | |
| 20. The Mediterranean <i>lies</i> between Europe and Africa. | |

We may speak of *laying* something or *setting* something, or may say that something is *laid* or is *set*; but we cannot speak of *lying* or *sitting* something, or of something being *lain* or *sat*. *Set*, in some of its meanings, is used without an object; as, "The sun *set*;" "He *set* out on a journey." *Set* is generally transitive; *sit*, always intransitive. *Lay* is transitive; *lie*, intransitive.

Lay, the present of the first verb, and *lay*, the past of *lie*, may easily be distinguished by the difference in meaning and in the time expressed.

Correct the following errors : —

Those things *have* not *came* to-day.

Wrong; because the past *came* is here used for the past participle *come*. The present perfect tense is formed by prefixing *have* to the past participle.

I done all my work before breakfast.

I come in a little late yesterday.

He has went to my desk without permission

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

That stupid fellow set down on my new hat.
He sat the chair in the corner.
Sit that plate on the table and let it set.
I have set in this position a long time.
That child will not lay still or set still a minute.
I laid down under the tree and enjoyed the scenery.
Lie that stick on the table and let it lay.
Those boys were drove out of the fort three times.
I have rode through the park.
I done what I could.
He has not spoke to-day.
The leaves have fell from the trees.
This sentence is wrote badly.
He throwed his pen down and said that the point was broke.
He teached me grammar.
I seen him when he done it.
My hat was took off my head and throwed out of the window.
The bird has flew into that tall tree.
I was chose leader.
I have began to do better. I begun this morning.
My breakfast was ate in a hurry.
Your dress sets well.
That foolish old hen is setting on a wooden egg.
He has tore it up and throwed it away.
William has took my knife, and I am afraid he has stole it.
This should be well shook.
I begun to sing before I knowed what I was doing.
We drunk from a pure spring.
I thought you had forsook us.
His pencil is nearly wore up.
He come and tell me all he knowed about it.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

218

Graded Lessons in English.

LESSON 97.

REVIEW QUESTIONS.

To the Teacher. — See "Scheme," p. 269.

How many modifications have verbs? *Ans.* — Five; viz., voice, mode, tense, number, and person. Define voice. How many voices are there? What verbs have voice? Define each. Illustrate. What is mode? How many modes are there? What mode is rejected by some? Define each. What is an infinitive? What is a participle? How many different kinds of participles are there? Define each. Illustrate. What is tense? How many tenses are there? Define each. Illustrate. What are the number and the person of a verb? Illustrate. What is conjugation? What is synopsis? What are auxiliaries? Name the auxiliaries. What are the principal parts of a verb? Why are they so called? How does a verb agree with its subject? When a verb has two or more subjects, how does it agree? Illustrate the uses of *shall* and *will*. Of *lie*, *lay*, *sit*, and *set*.

To the Teacher. — Select some of the preceding exercises, and require the pupils to write the parsing of all the verbs. See Lessons 34, 35, 48, 49, and 56.

Model for Written Parsing—Verbs. — *The Yankee, selling his farm, wanders away to seek new lands.*

CLASSIFICATION.		MODIFICATIONS.						SYNTAX.
<i>Verbs.</i>	<i>Kind.</i>	<i>Voice.</i>	<i>Mode.</i>	<i>Tense.</i>	<i>Num.</i>	<i>Per.</i>		
¹ selling	Pr. Par., Ir., Tr.	Ac.	Ind.	Pres.	Sing.	3d.		Mod. of <i>Yankee</i> .
wanders	Reg., Inf.	—	—	—	—	—		Pred. of " "
¹ seek	Inf., Ir., Tr.	Ac.	—	"	—	—		Prin. word in phrase
								Mod. of <i>wanders</i> .

¹ Participles and Infinitives have no person or number.

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"

Composition.

219

LESSON 98.

COMPOSITION.

Participles¹ sometimes partake of the nature of the noun while they retain the nature of the verb.

Use each of these phrases in a sentence, and explain the nature of the word in **ing** : —

Model. — “ — *in building* a snow fort; ” “ They were engaged *in building* a snow fort. ” *Building*, like a noun, follows the preposition *in*, as the principal word in the phrase; and, like a verb, it takes the object complement *fort*.

— by foretelling storms. — by helping others. — on approaching the house. — in catching fish.

Use the following phrases as subjects : —

Walking in the garden —. His writing that letter —.
Breaking a promise —.

Use each of these phrases in a complex sentence, letting some of the dependent clauses modify as adjectives, and some as adverbs : —

— in sledges. — up the Hudson. — down the Rhine.
— through the Alps. — with snow and ice. — into New York Bay. — on the prairie. — at Saratoga.

Build a short sentence containing all the parts of speech.

¹ If different names for the words in **ing** that have an adjective use and for those that have a noun use are desired, retain **participle** for the first and assign **nounal verb** to the second. We suggest these distinguishing names in “ Higher Lessons,” and use them in the “ High School Grammar.”

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

220

Graded Lessons in English.

Expand the following simple sentence into twelve sentences : —

Astronomy teaches the size, form, nature, and motions of the sun, moon, and stars.

Contract the following awkward compound sentence into a neat simple sentence : —

Hannibal passed through Gaul, and then he crossed the Alps, and then came down into Italy, and then he defeated several Roman generals.

Change the following complex sentences to compound sentences : —

When he asked me the question, I answered him courteously.

Morse, the man who invented the telegraph, was a public benefactor.

When spring comes, the birds will return.

Contract the following complex sentences into simple sentences by changing the verb in the dependent clause to the form in **ing** : —

A ship which was gliding along the horizon attracted our attention.

I saw a man who was plowing a field.

When the shower had passed, we went on our way.

I heard that he wrote that article.

That he was a foreigner was well known.

I am not sure that he did it.

Every pupil who has an interest in this work will prepare for it.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Miscellaneous Errors.

221

Change the following compound sentences to complex sentences : —

Model. — Morning dawns, and the clouds disperse =

When morning dawns, the clouds disperse.

Avoid swearing ; it is a wicked habit.

Pearls are valuable, and they are found in oyster shells.

Dickens wrote David Copperfield, and he died in 1870.

Some animals are vertebrates, and they have a backbone.

Expand each of the following sentences as much as you properly may : —

Indians dance. The clock struck. The world moves.

LESSON 99.

MISCELLANEOUS ERRORS.

Correct the following errors : —

I have got that book at home.

Wrong; because *have*, alone, asserts possession. *Got*, used in the sense of *obtained*, is correct; as, "I have just *got the book*."

Have you got time to help me?

There is many mistakes in my composition.

Wrong; because *is* should agree with its plural subject *mistakes*. The adverb *there* is often used to introduce a sentence, that the subject may follow the predicate. This often makes the sentence smooth and gives variety.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

222

Graded Lessons in English.

There goes my mother and sister.
Here comes the soldiers.
There was many friends to greet him.
It ain't there.

Ain't is a vulgar contraction. Correction — It *is not* there.

I have made up my mind that it ain't no use.
'Tain't so bad as you think.
Two years' interest were due.
Every one of his acts were criticised.
I, Henry, and you have been chosen.

Wrong; for politeness requires that you should mention the one spoken to, first; the one spoken of, next; and yourself, last.

He invited you and I and Mary.
Me and Jane are going to the fair.
I only want a little piece.
He is a handsome, tall man.
Did you sleep good?
How much trouble one has, don't they?
He inquired for some tinted ladies' note paper.
You needn't ask me nothing about it,
for I haven't got no time to answer.
Him that is diligent will succeed.
He found the place sooner than me.
Who was that? It was me and him.
If I was her, I would say less.
Bring me them tongs.
Us boys have a baseball club.
Whom did you say that it was?

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Analysis and Parsing.

223

Who did you speak to just now?
Who did you mean when you said that?
Where was you when I called?
There's twenty of us going.
Circumstances alters cases.
Tell them to set still.
He laid down by the fire.
She has lain her book aside.
It takes him everlastingly.
That was an elegant old rock.

LESSON 100.

ANALYSIS AND PARSING.

1. Thou shalt not take the name of the Lord thy God in vain.
2. Strike! till the last armed foe expires!
3. You wrong me, Brutus.
4. Shall we gather strength by irresolution and inaction?
5. Why stand we here idle?
6. Give me liberty or give me death!
7. Thy mercy, O Lord, is in the heavens, and thy faithfulness reacheth unto the clouds.
8. The clouds poured out water, the skies sent out a sound, the voice of thy thunder was in the heaven.
9. The heavens declare his righteousness, and all the people see his glory.
10. The verdant lawn, the shady grove, the variegated landscape, the boundless ocean, and the starry firmament are beautiful and magnificent objects.
11. When you grind your corn, give not the flour to the devil and the bran to God.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

224

Graded Lessons In English.

12. That which the fool does in the end the wise man does at the beginning.

13. Xerxes commanded the largest army that was ever brought into the field.

14. Without oxygen, fires would cease to burn, and all animals would immediately die.

15. Liquids, when acted upon by gravity, press downward, upward, and sideways.

16. Matter exists in three states—the solid state, the liquid state, and the gaseous state.

17. The blending of the seven prismatic colors produces white light.

18. Soap-bubbles, when they are exposed to light, exhibit colored rings.

19. He who yields to temptation debases himself with a debasement from which he can never arise.

20. Young eyes that last year smiled in ours
Now point the rifle's barrel ;
And hands then stained with fruits and flowers
Bear redder stains of quarrel.

CAPITAL LETTERS AND PUNCTUATION.

Capital Letters.—The first word of (1) a sentence, (2) a line of poetry, (3) a direct quotation making complete sense and a direct question introduced into a sentence and (4) phrases or clauses separately numbered or paragraphed should begin with a capital letter. Begin with a capital letter (5) proper names and words derived from them, (6) names of things personified, and (7) most abbreviations.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

viations. Write in capital letters (8) the words *I* and *O*, and (9) numbers in the Roman notation.¹

Examples. — 1. The judicious are always a minority.

2. Honor and shame from no condition rise ;

Act well your part, there all the honor lies.

3. The question is, "Can law make people honest?" 4. Paintings are useful for these reasons : 1. They please ; 2. They instruct.
5. The heroic Nelson destroyed the French fleet in Aboukir Bay.
6. Next, Anger rushed, his eyes on fire. 7. The Atlantic ocean beat Mrs. Partington. 8. The use of *O* and *oh* I am now to explain
9. Napoleon II. never came to the throne.

Period. — Place a period after (1) a declarative and an imperative sentence, (2) an abbreviation, and (3) a number written in the Roman notation.

For examples see 1, 7, and 9 above.

Interrogation Point. — Every direct interrogative sentence or clause should be followed by an interrogation point.

Example. — King Agrippa, believest thou the prophets ?

Exclamation Point. — All exclamatory expressions must be followed by the exclamation point.

Example. — Oh ! bloodiest picture in the book of time !

¹ Smaller letters are preferred where numerous references to chapters, etc., are made.

Comma. — Set off by the comma (1) a phrase out of its natural order or not closely connected with the word it modifies; (2) an explanatory modifier that does not restrict the modified term or combine closely with it; (3) a participle used as an adjective modifier, with the words belonging to it, unless restrictive; (4) the adjective clause when not restrictive; (5) the adverb clause unless it closely follows and restricts the word it modifies; (6) a word or phrase independent or nearly so; (7) a direct quotation introduced into a sentence, unless formally introduced; (8) a noun clause used as an attribute complement; and (9) a term connected to another by *or* and having the same meaning. Separate by the comma (10) connected words and phrases unless all the conjunctions are expressed; (11) independent clauses when short and closely connected; and (12) the parts of a compound predicate and of other phrases when long or differently modified.

Examples. — 1. In the distance, icebergs look like masses of burnished metal. 2. Alexandria, the capital of Lower Egypt, is an ill-looking city. 3. Labor, diving deep into the earth, brings up long-hidden stores of coal. 4. The sun, which is the center of our system, is millions of miles from us. 5. When beggars die, there are no comets seen. 6. Gentlemen, this, then, is your verdict. 7. God said, "Let there be light." 8. Nelson's signal was, "England expects every man to do his duty." 9. Rubbers, or overshoes, are worn to keep the feet dry. 10. The sable, the seal, and the otter furnish us rich furs. 11. His dark eye flashed, his proud breast heaved, his cheek's hue came and went. 12. Flights of birds darken the air, and tempt the traveler with the promise of abundant provisions.

Semicolon. — Independent clauses (1) when slightly connected, or (2) when themselves divided by the comma, must be separated by the semicolon. Use the semicolon (3) between serial phrases or clauses having a common dependence on something that precedes or follows; and (4) before *as*, *viz.*, *to wit*, *namely*, *i.e.*, and *that is*, when they introduce examples or illustrations.

Examples. — 1. The furnace blazes; the anvil rings; the busy wheels whirl round. 2. As Cæsar loved me, I weep for him; as he was fortunate, I rejoice at it; as he was valiant, I honor him; but, as he was ambitious, I slew him. 3. He drew a picture of the sufferings of our Saviour; his trial before Pilate; his ascent of Calvary; his crucifixion and death. 4. Gibbon writes, "I have been sorely afflicted with gout in the hand; to wit, laziness."

Colon. — Use the colon (1) between the parts of a sentence when these parts are themselves divided by the semicolon; and (2) before a quotation or an enumeration of particulars when formally introduced.

Examples. — 1. Canning's features were handsome; his eye, though deeply ensconced under his eyebrows, was full of sparkle and gayety: the features of Brougham were harsh in the extreme. 2. To Lentulus and Gellius bear this message: "Their graves are measured."

Dash. — Use the dash where there is an omission (1) of letters or figures, and (2) of such words as *as*, *namely*, or *that is*, introducing illustrations or equivalent expressions. Use the dash (3) where the sentence breaks off abruptly, and the same thought is resumed after a slight suspension,

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

228

Graded Lessons in English.

or another takes its place; and (4) before a word or phrase repeated at intervals for emphasis. The dash may be used (5) instead of marks of parenthesis, and may (6) follow other marks, adding to their force.

Examples. — 1. In *M——w*, ver. 3–11, you may find the “beatitudes.” 2. There are two things certain in this world—taxes and death. 3. I said—I know not what. 4. I never would lay down my arms—*never*—**NEVER**—**NEVER**. 5. Fulton started a steamboat—he called it the *Clermont*—on the Hudson in 1807. 6. My dear Sir,—I write this letter for information.

Marks of Parenthesis. — Marks of parenthesis may be used to inclose what has no essential connection with the rest of the sentence.

Example. — The noun (Lat. *nomen*, a name) is the first part of speech.

Apostrophe. — Use the apostrophe (1) to mark the omission of letters, (2) in the pluralizing of letters, figures, and characters, and (3) to distinguish the possessive from other cases.

Examples. — 1. Bo’t of John Jones 10 lbs. of butter. 2. What word is there one-half of which is *p’s* ? 3. He washed the disciples’ feet.

Hyphen. — Use the hyphen (-) (1) between the parts of compound words that have not become consolidated, and (2) between syllables when a word is divided.

Examples. — 1. Work-baskets are convenient. 2. Divide *basket* thus: *bas-ket*.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Quotation Marks. — Use quotation marks to inclose a copied word or passage. If the quotation contains a quotation, the latter is inclosed within single marks.

Example. — The sermon closed with this sentence, "God said 'Let there be light.'"

Brackets. — Use brackets [] to inclose what, in quoting another's words, you insert by way of explanation or correction.

Example. — The Psalmist says, "I prevented [anticipated] the dawning of the morning."

LETTER-WRITING.

In writing a letter there are six things to consider — the **heading**, the **introduction**, the **body of the letter**, the **conclusion**, the **folding**, and the **superscription**.

THE HEADING.

Parts. — The Heading consists of the name of the **Place** at which the letter is written, and the **Date**. If you write from a city, give the door-number, the name of the street, the name of the city, and the name of the state. If you are at a hotel or a school or any other well-known institution, its name may take the place of the door-number and the name of the street. If you write from a village or other country place, give your post-office address, the name of the county, and that of the state.

The Date consists of the name of the month, the day of the month, and the year.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

230

Graded Lessons in English.

How Written. — Begin the Heading about an inch and a half from the top of the page — on the first ruled line of commercial note — and a little to the left of the middle of the page. If the Heading is very short, it may stand on one line. If it occupies more than one line, the second line should begin further to the right than the first, and the third further to the right than the second.

The Date stands upon a line by itself if the heading occupies two or more lines.

The door-number, the day of month, and the year are written in figures; the rest, in words. Each important word begins with a capital letter, each item is set off by the comma, and the whole closes with a period.

Study what has been said, and write the following headings according to these models :—

- | | |
|---|--|
| 1. Hull, Mass., Nov. 1, 1860 | 3. Newburyport, Mass.,
June 30, 1900. |
| 2. 1466 Colorado Ave.,
Rochester, N.Y.,
Apr. 3, 1870. | 4. Starkville, Herkimer Co., N.Y.,
Dec. 19, 1871. |
| 1. n y rondout 11 1849 oct. 2. staten island port richmond 1877
25 january. 3. brooklyn march 1871 mansion house 29. 4. execu-
tive chamber vt february montpelier 1869 27. 5. washington frank-
lin co mo nov 16 1874. 6. fifth ave may new york 460 9 1901.
7. washington d c march 1900 520 pennsylvania ave 16. | |

THE INTRODUCTION.

Parts. — The Introduction consists of the **Address** — the Name, the Title, and the Place of Business or the Resi-

Digitized by Google

dence of the one addressed — and the **Salutation**. Titles of respect and courtesy should appear in the Address. Prefix *Mr.* (plural, *Messrs.*) to a man's name; *Master* to a boy's name; *Miss* to the name of a girl or an unmarried lady; *Mrs.* to the name of a married lady. Prefix *Dr.* to the name of a physician, or write *M.D.* after his name. Prefix *Rev.* (or *The Rev.*) to the name of a clergyman; if he is a Doctor of Divinity, prefix *Rev. Dr.*, or write *Rev.* before his name and *D.D.* after it; if you do not know his Christian name, prefix *Rev. Mr.* or *Rev. Dr.* to his surname, but never *Rev.* alone. *Esq.* is added to the name of a lawyer, and to the names of other prominent men. Avoid such combinations as the following: *Mr. John Smith, Esq.*; *Dr. John Smith, M.D.*; *Mr. John Smith, M.D.*

Salutations vary with the station of the one addressed, or the writer's degree of intimacy with him. Strangers may be addressed as *Sir*, *Rev. Sir*, *General*, *Madam*, *Miss Brown*, etc.; acquaintances as *Dear Sir*, *Dear Madam*, etc.; friends as *My dear Sir*, *My dear Madam*, *My dear Mr. Brown*, etc.; and near relatives and other dear friends as *My dear Wife*, *My dear Boy*, *Dearest Ellen*, etc.

How Written. — The Address may follow the Heading, beginning on the next line or the next but one, and standing on the left side of the page; or it may stand in corresponding position after the Body of the Letter and the Conclusion. If the letter is written to a very intimate friend, the Address may appropriately be placed at the

bottom of the letter ; but in other letters, especially those on ordinary business, it should be placed at the top and as directed above. There should always be a narrow margin on the left-hand side of the page, and the Address should always begin on the marginal line. If the Address occupies more than one line, the initial words of these lines should slope to the right as in the Heading.

Begin the Salutation on the marginal line or a little to the right of it, when the Address occupies three lines ; on the marginal line or further to the right than the second line of the Address begins, when this occupies two lines ; a little to the right of the marginal line, when the Address occupies one line ; on the marginal line, when the Address stands below.

Every important word in the Address should begin with a capital letter. All the items of it should be set off by the comma ; and, as it is an abbreviated sentence, it should close with a period. Every important word in the Salutation should begin with a capital letter, and the whole should be followed by a comma.

Study what has been said, and write the following introductions according to these models : —

- | | |
|----------------------------------|---------------------------|
| 1. Dear Father, | 3. Messrs. Clark & Brown, |
| I write, etc. | Quogue, N.Y. |
| 2. The Rev. M. H. Buckham, D.D., | Gentlemen, |
| President of U. V. M., | 4. Messrs. Tiffany & Co., |
| Burlington, Vt. | 2 Milk St., Boston. |
| My dear Sir, | Dear Sirs, |

1. henry s snow lld president of polytechnic institute brooklyn n y
dear sir. 2. dr john h hobart burge 64 livingston st brooklyn n y
sir. 3. prof geo n boardman chicago ill dear teacher. 4. to the
president executive mansion washington d c mr president. 5. rev t
k bunker elmira n y sir. 6. messrs gilbert & sons gentlemen mass
boston. 7. mr george r curtis minn rochester my friend dear. 8. to
the honorable john hay secretary of state washington d c sir.

THE BODY OF THE LETTER.

The Beginning. — Begin the Body of the Letter at the end of the Salutation, and on the same line if the Introduction consists of four lines — in which case the comma after the Salutation should be followed by a dash ; otherwise, on the line below.

Style. — Be perspicuous. Paragraph and punctuate as in other kinds of writing. Spell correctly ; write legibly, neatly, and with care. A letter tells a great deal of the writer — more, oftentimes, than the writer means to say or supposes that he is saying.

Letters of friendship should be natural, familiar, and colloquial. Whatever is interesting to you will be interesting to your friends.

Business letters should be brief, and the sentences should be short, concise, and to the point.

In formal notes the third person is generally used instead of the first and second ; there is no Introduction, no Conclusion, no Signature, only the name of the Place and the Date at the bottom, on the left side of the page.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

234

Graded Lessons in English.

THE CONCLUSION.

Parts.—The Conclusion consists of the **Complimentary Close** and the **Signature**. The forms of the Complimentary Close are many, and are determined by the relations of the writer to the one addressed. In letters of friendship, you may use *Your sincere friend*; *Yours affectionately*; *Your loving son or daughter*, etc. In business letters, you may use *Yours*; *Yours truly*; *Truly yours*; *Yours respectfully*; *Very respectfully yours*, etc. In official letters, use *I have the honor to be, Sir, your obedient servant*; *Very respectfully, your most obedient servant*.

The Signature consists of your Christian name and your surname. In addressing a stranger write your Christian name in full. A lady addressing a stranger should prefix her title—*Miss* or *Mrs.*—to her own name, enclosing it within marks of parenthesis if she wishes.

How Written.—The Conclusion should begin near the middle of the first line below the Body of the Letter, and should slope to the right like the Heading and the Address. Begin each line of it with a capital letter, and punctuate as in other writing, following the whole with a period. The Signature should be very plain.

THE FOLDING.

The Folding is a simple matter when, as now, the envelope used is adapted in length to the width of the sheet. Take the letter as it lies before you, with its first page

Digitized by Google

uppermost, turn up the bottom of it about one-third the length of the sheet, bring the top down over this, taking care that the sides are even, and press the parts together. Taking the envelope with its back toward you, insert the letter, putting in first the edge last folded.

The form of the envelope may require the letter to be folded in the middle. Other conditions may require other ways of folding.

THE SUPERScription.

Parts. — The Superscription is what is written on the outside of the envelope. It is the same as the Address, consisting of the Name, the Title, and the full Directions of the one addressed.

How Written. — The Superscription should begin near the middle of the envelope and near the left edge — the envelope lying with its closed side toward you — and should occupy three or four lines. These lines should slope to the right as in the Heading and the Address, the spaces between the lines should be the same, and the last line should end near the lower right-hand corner. On the first line the Name and the Title should stand. If the one addressed is in a city, the door-number and name of the street should be on the second line, the name of the city on the third, and the name of the state on the fourth. If he is in the country, the name of the post office should be on the second line, the name of the county on the third (or by itself near the lower left-hand corner), and the

name of the state on the fourth. The titles following the name should be separated from it and from each other by the comma, and every line should end with a comma, except the last, which should be followed by a period. The lines should be straight, and every part of the Superscription should be legible. Place the stamp at the upper right-hand corner.

We give, on succeeding pages, a few letters illustrating the various forms used.

LETTER, ORDERING MERCHANDISE.

Newburgh, N.Y.,
Jan. 7, 1899.

Messrs. Hyde & Co.,
250 Broadway, N.Y.
Gentlemen,

Please send me by
Adams Express the articles men-
tioned in the enclosed list.

Be careful in the selection of
the goods, as I desire them for a
special class of customers.

When they are forwarded,
please inform me by letter and
enclose the invoice.

Yours truly,
Thomas Dodds.

238

Graded Lessons in English.

ANSWER, INCLOSING INVOICE.

250 Broadway, N.Y.
Jan. 9, 1899.Mr. Thomas Dodds,
Newburgh, N.Y.

Dear Sir,

We have to-day sent you by
Adams Express the goods ordered
in your letter of the 7th inst.
Enclosed you will find the invoice.

We hope that everything will
reach you in good condition and
will prove satisfactory in qual-
ity and in price.

Very truly yours,
Peter Hyde & Co.

Thomas Dodds,

INVOICE.

Bought of Peter Hyde & Co.

3 boxes Sperm Candles, 140 lbs.,	@ 33c.	\$46 20
7 do. Adamantine Extra Candles, 182 lbs.,	" 26c.	47 32
120 lbs. Crushed Sugar,	" 12½c.	15 00
60 do. Coffee do.,	" 11½c.	6 75
		<u>\$115.27</u>

Digitized by Google

LETTER OF APPLICATION.

176 Clinton St., Brooklyn, N.Y.,
Dec. 12, 1899.

Messrs. Fisk & Hatch,
5 Nassau St., N.Y.

Gentlemen,

Learning by advertisement that a clerkship in your house is vacant, I beg leave to offer myself as a candidate for the place.

I am sixteen years old, and am strong and in excellent health. I have just graduated with honor from the seventh grade of the Polytechnic Institute, Brooklyn, and I enclose testimonials of my character and standing from the President of that Institution.

If you desire a personal interview, I shall be glad to present myself at such time and place as you may name.

Very respectfully yours,
Charles Hastings.

NOTES OF INVITATION AND ACCEPTANCE

(IN THE THIRD PERSON).

Mr. and Mrs. Brooks request the pleasure of Mr. Churchill's company at a social gathering, next Tuesday evening, at 8 o'clock. *Mr. Churchill has much pleasure in accepting Mr. and Mrs. Brooks's kind invitation to a social gathering, next Tuesday evening.*

32 W. 31st Street, Oct. 5.

160 Fifth Ave., Oct. 5.

LETTER OF INTRODUCTION.

Concord, N. H.,
Jan. 28, 1899.
George Chapman, Esq.,
Portland, Conn.
My dear Friend,

It gives me great pleasure to introduce to you my friend, Mr. Alpheus Crane. Any attentions you may be able to show him I shall esteem as a personal favor.

Sincerely yours,
Peter Cooper.

A LETTER OF FRIENDSHIP.

21 Dean St., Toledo, Ohio,
Dec. 16, 1899.

My dear Mother,

I cannot tell
you how I long to be at home
again and in my old place.
In my dreams and in my
waking hours, I am often
back at the old homestead;
my thoughts play truant
while I pore over my books,
and even while I listen to my
teacher in the class-room. I
would give so much to know
what you all are doing — so
much to feel that now and
then I am in your thoughts,
and that you do indeed "miss

me at home."

Everything here is as pleasant as it need be or can be, I suppose. I am sure I shall enjoy it all by and by, when I get over this fit of homesickness.

My studies are not too hard, and my teachers are kind and faithful.

Do write me a long letter as soon as you get this, and tell me everything.

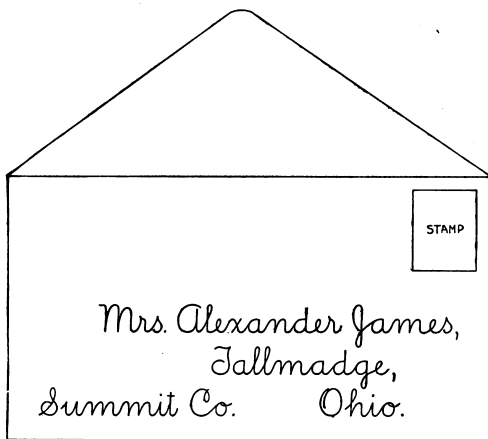
Much love to each of the dear ones at home.

Your affectionate son,

Henry James.

¹Mrs. Alexander James,
Tallmadge, Ohio.

¹ In familiar (and official) letters, the Address may stand, you will remember, at the bottom.



To the Teacher. — Have your pupils write complete letters and notes of all kinds. You can name the persons to whom these are to be addressed. Attend minutely to all the points. Letters of introduction should have the word *Introducing* (followed by the name of the one introduced) at the lower left-hand corner of the envelope. This letter should not be sealed. The receiver may seal it before handing it to the one addressed.

Continue this work of letter-writing until the pupils have mastered all the details, and are able easily and quickly to write any ordinary letter.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

244

Graded Lessons In English.

A SUMMARY OF THE RULES OF SYNTAX.

I. A noun or pronoun used as subject or as attribute complement of a predicate verb, or used independently, is in the nominative case.

II. The attribute complement of a participle or an infinitive is in the same case (nominative or objective) as the word to which it relates.

III. A noun or pronoun used as possessive modifier is in the possessive case.

IV. A noun or pronoun used as object or objective complement, or as the principal word of a prepositional phrase, is in the objective case.

V. A noun or pronoun used as explanatory modifier is in the same case as the word explained.

VI. A pronoun agrees with its antecedent in person, number, and gender.

With two or more antecedents connected by *and*, the pronoun is plural.

With two or more singular antecedents connected by *or* or *nor*, the pronoun is singular.

VII. A verb agrees with its subject in person and number.

With two or more subjects connected by *and*, the verb is plural.

With two or more singular subjects connected by *or* or *nor*, the verb is singular.

VIII. A participle assumes the action or being, and is used like an adjective or a noun.

IX. An infinitive is generally introduced by *to*, and with it forms a phrase used as a noun, an adjective, or an adverb.

X. Adjectives modify nouns or pronouns.

XI. Adverbs modify verbs, adjectives, or adverbs.

XII. A preposition introduces a phrase modifier, and shows the relation, in sense, of its principal word to the word modified.

XIII. Conjunctions connect words, phrases, or clauses.

XIV. Interjections are used independently.

Digitized by Google

PROOF-MARKS.

Remark.—The following are some of the marks used in correcting proof-sheets for the printer :—

- § Dē-le = Strike out.
- ^ Cā-ret = Something to be inserted.
- | This calls attention to points or letters placed in the margin as corrections.
- ⊙ This calls attention to the period.
- tr. Transpose.
- ¶ Begin a new paragraph with the word preceded by [.
- No ¶ No new paragraph.
- ∨ This calls attention to the apostrophe.

To the Teacher.—We suggest that the pupils learn to use these marks in correcting compositions. The following exercises are given as illustrations :—

§ ⊙ Capt. James B. Eads,
 ⊙ M St. Louis, Mo.
 ¶ ⊙ Hon. Andrew D. White, LL.D.,
 ¶ ⊙ Ithaca, N.Y.
 § ¶ Miss Kate Field,
 L C Salt Lake City,
 Utah.

C. | Ocala, Marion Co., Fla.,
 Jan. 10, '99.

d | My Dear Friend,

Yours of the

21st Inst. was welcome.

No 7 (How I enjoyed the story
 of your Christmas vacation!

Joel Chandler Harris's

You are an excellent letter-
 & writer. [My vacation was spent
 quietly, but with "St. Nicholas,"
 "The Youth's Companion," and
 "Nights with Uncle Remus" one
 tr. could be hardly dull.

Very sincerely yours,

David Copperfield.

Master

Samuel Gulliver,
 San Diego, Cal.

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"

REVIEW OF GRADED LESSONS.

WORDS—SPOKEN AND WRITTEN.

Spoken words are composed of **sounds**. Written words are composed of **letters** called (1) **vowels**—**a, e, i, o,** and **u**—representing the open sounds, and (2) **consonants** representing (*a*) obstructed breath vocalized; **as, b, d, g,** etc., called **sonants**, and (*b*) obstructed breath unvocalized; **as, p, t, k,** etc., called **surds**.

Spoken and written words form **verbal language**; and tones, gestures, and facial expression form **natural language**—used to reinforce spoken.

DEFINITION.—**English grammar** is the science which teaches the forms, uses, and relations of the words of the English language.

Language is used for the purpose of communicating thought, and the unit of thought, and of expression therefore, is

A SENTENCE.

DEFINITION.—A **sentence** is a group of words expressing a thought.

Its two parts are **subject** and **predicate**.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

248

Graded Lessons In English.

DEFINITIONS.

The **subject** of a sentence names that of which something is thought.

The **predicate** of a sentence tells what is thought.

A **phrase** is a group of words denoting related ideas but not expressing a thought.

A **clause** is a part of a sentence containing a subject and its predicate.

A **modifier** is a word or a group of words joined to some part of a sentence to qualify or limit the meaning.

The subject with its modifiers is called the **modified subject**; and the predicate with its modifiers is called the **modified predicate**.

Greece, which is the most noted country of antiquity, scarcely exceeded in size and in population the half of the state of New York.

The whole is a **sentence**; *Greece* is **subject**, *exceeded* is **predicate**; *Greece . . . antiquity* is the **modified subject**, *scarcely . . . New York* is the **modified predicate**; *which . . . antiquity* is a **clause**; *noted* and *scarcely* are **simple word modifiers**; *of antiquity* is a **simple phrase modifier**; *in size and in population* is a **compound phrase modifier**; *of the state of New York* is a **complex phrase modifier** — the phrase of *New York* modifying *state*, a word in the phrase of *the state* — and *which . . . antiquity* is a **clause modifier** of *Greece*.

DEFINITIONS.

The **analysis** of a sentence is the separation of it into its parts.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

A **diagram** is a picture of the offices and relations of the different parts of a sentence.

Synthesis, **construction**, or **composition** is the putting together (1) of words, phrases, and clauses to form **sentences**, (2) of sentences to form a **paragraph**, and (3) of paragraphs to form a **theme**.

We group words into classes with respect to their office in the sentence. These classes, eight in number and called *parts of speech*, are the **noun**, the **pronoun**, the **verb**, the **adjective**, the **adverb**, the **preposition**, the **conjunction**, and the **interjection**.

The first five of these undergo what are called **modifications** — changes in form, meaning, and use.

CLAUSES CLASSIFIED.

He *that runs* may read it ; He may read it *if he will keep the fact secret* ; It is true *that he read it*.

In each of these sentences there are two clauses and two kinds of clauses. Those not italicized are **independent clauses** ; those italicized are **dependent clauses** — the first an **adjective clause**, the second an **adverb clause of condition**, and the last a **noun clause explanatory**.

DEFINITIONS.

A **dependent clause** is one used as an adjective, an adverb, or a noun.

An **independent clause** is one not dependent on another clause.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

250

Graded Lessons in English.

SENTENCES CLASSIFIED.

Knowledge comes ; Knowledge comes, but wisdom lingers ; Knowledge comes, though wisdom lingers.

The first is a **simple sentence** ; the second is a **compound sentence**, made up of two independent clauses ; and the third is a **complex sentence**, made up of an independent and a dependent clause.

DEFINITIONS.

A **simple sentence** is one that contains but one subject and one predicate, either or both of which may be compound.

A **compound sentence** is one composed of two or more independent clauses.

A **complex sentence** is one composed of an independent clause and one or more dependent clauses.

John runs ; Does John run ? Run, John ; How John runs !

The first sentence utters a fact, the second asks a question, the third issues a command, and the fourth expresses sudden feeling.

DEFINITIONS.

A **declarative sentence** is one that is used to affirm or to deny.

An **interrogative sentence** is one that expresses a question.

An **imperative sentence** is one that expresses a command or an entreaty.

Digitized by Google

An **exclamatory sentence** is one that expresses sudden thought or strong feeling.

THE NOUN.

Mary's mother, the wife of the merchant, bought her daughter a house a few months ago ; My son, make wisdom the object of your life, for it is the principal thing.

The words italicized in these two sentences perform very different offices : (1) *mother* is **subject**, (2) *house* is **object**, (3) *Mary's* is a **possessive modifier** of *mother*, (4) *wife* is **explanatory** of *mother*, (5) *merchant* is **chief word** in a **prepositional phrase**, (6) *daughter* is **indirect object** of an action, (7) *months* has an **adverbial use measuring time**, (8) *son* is **independent** by address, (9) *object* is **objective complement**, and (10) *thing* is **attribute complement**. But, while discharging each a special function, they all have one function — they **name** persons or things, and hence are called **nouns**.

DEFINITION. — A **noun** is a name of anything.

There are two kinds of nouns — those naming all things of a certain class, and hence called **common nouns**, and those that are each the particular name of an individual of a class, and hence called **proper nouns**.

DEFINITIONS.

A **common noun** is a name which belongs to all things of a class.

A **proper noun** is the particular name of an individual.

Nouns have four **modifications** — **number, gender, person, and case**.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

252

Graded Lessons in English.

NUMBER.

DEFINITIONS.

Number is that modification of a noun or pronoun which denotes one thing or more than one.

The **singular number** denotes one thing.

The **plural number** denotes more than one thing.

RULE. — The **plural** of nouns is regularly formed by adding **s** or **es** to the singular.

The **s** is a more common plural ending than the **es**.

The **es** is added (1) to words ending in **s**, **x**, **z**, **sh**, and **ch**, and makes a separate syllable, as in **gases**, **foxes**, **topazes**, **lashes**, and **birches**; (2) to many nouns in **o**, as in **cargoes**, **negroes**, and **mottoes**; (3) to nouns in **y**, the **y** when preceded by a consonant changing to **i**, as in **cities**, **daisies**, and **skies**; and (4) to some nouns in **f** or **fe**, the **f** or **fe** changing to **v**, as in **loaves**, **calves**, **lives**, and **knives**.

Some nouns form their plural **irregularly**, (1) by **internal change**, as in the six nouns, **man**, **men**; **foot**, **feet**; **tooth**, **teeth**; **goose**, **geese**; **louse**, **lice**; and **mouse**, **mice**; (2) by adding **en**, as in **ox**, **oxen**; **child**, **children**; and (3) by keeping the singular form, as in **deer**, **deer**; and **sheep**, **sheep**.

GENDER.

DEFINITIONS.

Gender is that modification of a noun or pronoun which distinguishes sex.

The **masculine gender** denotes the male sex.

The **feminine gender** denotes the female sex.

The **neuter gender** denotes want of sex.

Gender in English follows the sex of the object named.

Digitized by Google

Strictly speaking, there can be but two genders, as there can be but two sexes—the names of objects without sex are of the **neuter** (neither) gender, therefore.

The three ways of distinguishing the feminine from the masculine are (1) by a **change of ending**, as in host, **hostess**; and Jew, **Jewess**; (2) by a change of a word in the name, as in man-servant, **maid-servant**; gentleman, **gentlewoman**; and peacock, **peahen**; and (3) by the use of words wholly or radically different; as, boy, **girl**; lord, **lady**; and wizard, **witch**.

PERSON.

Number and gender are modifications of nouns affecting the meaning—number almost always indicated by the ending, gender sometimes.

Person is a modification of nouns that is not accompanied by form, as in

I Paul have written; Paul, thou art beside thyself; He brought Paul before Agrippa.

Paul retains its form, though in the first it names the speaker, and is of the **first** person; in the second it names the one spoken to, and is of the **second** person; and in the third it names the one spoken of, and is in the **third** person.

DEFINITIONS.

Person is that modification of a noun or pronoun which denotes the speaker, the one spoken to, or the one spoken of.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

254

Graded Lessons in English.

The **first person** denotes the one speaking.
The **second person** denotes the one spoken to.
The **third person** denotes the one spoken of.

CASE.

The *bear* killed the man; The *man* killed the *bear*; *Bear's* grease is made into hair oil.

In 1 the bear is represented as performing an action; in 2 as receiving an action; in 3 as possessing something. The word *bear* in these sentences has three different **uses** and is in the three cases — **nominative** in 1, **objective** in 2, and **possessive** in 3 — only the possessive being indicated by form.

In the illustrative sentences on p. 251, (1) the subject *mother*, wife explanatory of *mother*, son independent by address, and the attribute complement *thing* are all in the **nominative case**; (2) *Mary's*, possessive modifier of *mother*, is in the **possessive case**; and (3) *merchant*, principal word in a prepositional phrase, *daughter*, indirect object of an action, the object complement *house*, *months*, adverbial to denote measure, and *object*, the objective complement of *make*, are all in the **objective case**.

DEFINITIONS.

The **attribute complement** completes the predicate and belongs to the subject.

The **object complement** completes the predicate and names that which receives the act.

The **objective complement** completes the predicate and belongs to the object complement.

Case is that modification of a noun or pronoun which denotes its office in the sentence.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Review of Graded Lessons.

255

The **nominative case** of a noun or pronoun denotes its office as subject or as attribute complement.

The **possessive case** of a noun or pronoun denotes its office as possessive modifier.

The **objective case** of a noun or pronoun denotes its office as object complement, or as principal word in a prepositional phrase.

(The definitions of the nominative and objective cases give only their principal offices.)

DEFINITION. — **Declension** is the arrangement of the cases of nouns and pronouns in the two numbers.

DECLENSION OF NOUNS.

Nom. boy, boys, lady, ladies, man, men,

Pos. boy's, boys', lady's, ladies', man's, men's,

Obj. boy; boys. lady; ladies. man; men.

RULE. — The **possessive case** of nouns is formed in the singular by adding to the nominative the apostrophe and the letter **s** (**'s**); in the plural, by adding (**'**) only. If the plural does not end in **s**, the apostrophe and the **s** are both added.

The preposition *of* and the objective may be used in place of the possessive—the wing *of the fly*=the *fly's* wing.

The possessive sign is added (1) to each of several nouns when modifying different words; as, Webster's and Worcester's dictionary; (2) to the last only, when modifying the same word; as, Ticknor & Field's bookstore.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

256

Graded Lessons in English.

THE PRONOUN.

DEFINITION. — A **pronoun** is a word used for a noun.

The word, phrase, or clause for which a pronoun stands is called its **antecedent**.

Pronouns have the modifications of nouns—**number**, **gender**, **person**, and **case**.

CLASSES.

Those that by their form denote the speaker, the one spoken to, and the one spoken of are called **personal pronouns**—*I*, of the **first** person ; *thou* and *you*, of the **second** person ; and *he*, *she*, and *it*, of the **third** person.

Those used in asking questions are called **interrogative pronouns**—*who*, *which*, and *what*.

Those that refer to some word or words in another clause and so connect clauses are called **relative pronouns**—*who*, *which*, *what*, and *that*.

Those used as adjectives and nouns—*all*, *some*, *both*, *many*, etc. — are called **adjective pronouns**.

DEFINITIONS.

A **personal pronoun** is one that by its form denotes the speaker, the one spoken to, or the one spoken of.

An **interrogative pronoun** is one with which a question is asked.

A **relative pronoun** is one that refers to some word or words in another clause and connects clauses.

Digitized by Google

An **adjective pronoun** is one that performs the offices of an adjective and a noun.

On pp. 186-188, we see (1) that personal, interrogative, and relative pronouns do not add *s* to form the plural ; (2) that no personal pronoun, except *you*, forms its plural from the singular ; (3) that no personal, interrogative, or relative pronoun has the apostrophe and *s* in the possessive singular or the apostrophe in the possessive plural ; (4) that every personal pronoun has two forms in the possessive plural, and that all but *he* and *it* have two forms in the possessive singular ; (5) that *he* is always masculine, *she* feminine, and *it* neuter, and that *I*, *you*, and *thou* are of any gender ; (6) that *he*, *she*, and *it* have the same plural, and therefore *they*, *their*, and *them* are of any gender ; (7) that *self* added to the possessives *my*, *thy*, and *your*, and to the objectives *him*, *her*, or *it*, makes our **compound personal pronouns** in the singular ; (8) that *selves* added to *our* and *your* and to *them* makes the same pronouns in the plural ; (9) that *who* and *which* have their plurals like the singular ; (10) that *what* and *that* are indeclinable ; (11) that *whose* is the possessive of the interrogative and the relative *who* and *which* ; (12) that, excepting *that*, the relatives and the interrogatives are the same ; (13) that *ever* and *soever* added to *who*, *which*, and *what* form our **compound relative pronouns** ; (14) that the relative *who* represents persons ; *which*, animals and things ; *that*, persons, animals, and things ; and *what*, things ;

and (15) that the **only nominative** and **objective forms** in English — **eight** of one and **seven** of the other — are in the declensions of these three classes of pronouns.

Pronouns agree with their antecedents in number, gender, and case.

THE ADJECTIVE.

DEFINITION. — An **adjective** is a word used to modify a noun or a pronoun.

Good men ; *six* marbles ; *much* land ; *this* book.

Good denotes quality ; *six*, number ; *much*, quantity ; and *this*, the relation of the book to the speaker.

CLASSES OF ADJECTIVES.

DEFINITIONS.

A **descriptive adjective** is one that modifies by expressing quality.

A **definitive adjective** is one that modifies by pointing out, numbering, or denoting quantity.

In "A *wise*, *capable*, and *influential* teacher is *simple* and *unaffected* in speech and in bearing," we see that adjectives (1) may be **assumed**, and (2) may be **asserted**—standing in the predicate as **attribute complements**. We see (3) their **punctuation**, and (4) in **what order** they stand when of different length.

In "A *wise* man is respected," we may for the adjective *wise* substitute (5) the **equivalent phrase of wisdom** or (6) the **equivalent clause** *who is wise*, and thus secure **variety** of expression.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Care is needed in selecting **apt** adjectives, and in guarding against an **excessive** use of them.

COMPARISON.

Adjectives have one modification — **comparison** — seen in lovely, lovelier, loveliest ; lovely, **more** lovely, **most** lovely ; lovely, **less** lovely, **least** lovely. The terminations **er** and **est** and the prefixed adverbs **more** and **most** denote **increase** of the quality ; the prefixed adverbs **less** and **least** denote **diminution**.

DEFINITIONS.

Comparison is a modification of the adjective to express the relative degree of the quality in the things compared.

The **positive degree** expresses the simple quality.

The **comparative degree** expresses a greater or a less degree of the quality.

The **superlative degree** expresses the greatest or the least degree of the quality.

If we suppose that in comparing we express increase oftener than decrease, and increase oftener by **er** and **est** than by **more** and **most**, we have the

RULE. — Adjectives are regularly compared by adding **er** to the positive to form the comparative, and **est** to the positive to form the superlative.

THE ADVERB.

DEFINITION. — An **adverb** is a word used to modify a verb, an adjective, or an adverb.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

260

Graded Lessons in English.

CLASSES OF ADVERBS.

Those that answer the question, *When?* are **adverbs of time**.

Those that answer the question, *Where?* are **adverbs of place**.

Those that answer the question, *To what extent?* are **adverbs of degree**.

Those that answer the question, *In what way?* are **adverbs of manner**.

Adverbs that connect clauses and modify words in them are called **conjunctive adverbs**.

In "We started *then*," we may substitute for the adverb *then* the phrase *at that time* or the clause *when the time came*.

Adverbs, then, may be expanded into equivalent phrases and clauses, and such phrases and clauses may be contracted into equivalent adverbs.

Adverbs, like adjectives, are compared. For the lists of adjectives and adverbs compared irregularly, see Lessons 87 and 88. In these lists it is seen that *more* and *most*, *less* and *least*, used in comparing adjectives and adverbs, are themselves comparatives and superlatives in *er* and *est* slightly disguised—see "Higher Lessons," Revised Edition, p. 259, foot-note.

Care is needed in the choice of adverbs, and in placing them and adverbial phrases where they belong.

THE VERB.

DEFINITION.—A **verb** is a word that asserts action, being, or state of being.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

It **asserts**, whether the sentence affirms, denies, or questions.

CLASSES OF VERBS.

The boy *caught* a fish ; Fish *swim*. *Caught* needs an object complement, as *fish*, to make a complete assertion ; *swim* does not. *Caught* and all verbs that denote an act as going over from a doer to a receiver are **transitive** ; *swim* and all verbs that do not require a word to complete the assertion are **intransitive**.

DEFINITIONS.

A **transitive verb** is one that requires an object.

An **intransitive verb** is one that does not require an object.

I *crush* the worm ; I *crushed* the worm ; The worm *crushed* by me died.

I *drive* the horses ; I *drove* the horses ; The horses *driven* by me ran away.

The past tense and the past participle *crushed* is formed by adding **ed** to the present *crush* ; the past tense *drove* is formed by **vowel-change** of the present *drive* ; and the past participle is formed by adding **en**. *Crush* and verbs like it are **regular** ;¹ *drive* and those like it are **irregular**.¹

DEFINITIONS.

A **regular verb** is one that forms its past tense and past participle by adding **ed** to the present.

An **irregular verb** is one that does not form its past tense and past participle by adding **ed** to the present.

¹ For another classification, see Lesson 74, foot-note.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

262

Graded Lessons in English.

Verbs have the modifications called **voice**, **mode**, **tense**, **number**, and **person**.

VOICE.

I *drove* the horses ; The horses *were driven* by me.

Drove shows that the subject denotes the actor ; *were driven* shows that the subject names the ones acted upon. These uses of the verb constitute the modification called **voice** : *drove* is in the **active** voice, and *were driven* is in the **passive**.

DEFINITIONS.

Voice is that modification of the transitive verb which shows whether the subject names the actor or the thing acted upon.

The **active voice** shows that the subject names the actor.

The **passive voice** shows that the subject names the thing acted upon.

MODE.

James *walks* ; James *may walk* ; If James *walk* out, he will improve ; James, *walk* on.

Here the action is asserted (1) as a fact, (2) as possible, (3) as conceivable, and (4) as a command ; and these ways of asserting give us the four modes — **indicative**, **potential**, **subjunctive**, and **imperative**.
Lesson 90, foot-note.

DEFINITIONS.

Mode is that modification of the verb which denotes the manner of asserting the action or being.

The **indicative mode** asserts the action or being as a fact.

The **potential mode** asserts the power, liberty, possibility, or necessity of acting or being.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

The **subjunctive mode** asserts the action or being as a mere supposition, conception, or wish.

The **imperative mode** asserts the action or being as a command or an entreaty.

TENSE.

I walk; I walked; I shall walk; I have walked; I had walked; I shall have walked.

In the first three sentences, the action is asserted as **taking place** in time (1) present, (2) past, and (3) future; in the last three it is asserted as finished or **completed** in time (4) present, (5) past, and (6) future.

DEFINITIONS.

Tense is that modification of the verb which expresses the time of the action or being.

The **present tense** expresses action or being as present.

The **past tense** expresses action or being as past.

The **future tense** expresses action or being as yet to come.

The **present perfect tense** expresses action or being as completed at the present time.

The **past perfect tense** expresses action or being as completed at some past time.

The **future perfect tense** expresses action or being to be completed at some future time.

PERSON AND NUMBER.

I walk; Thou walkest; He walks; They walk. Walk adds the ending **est** and **s** in the second and third sentences to make the verb

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

216

Graded Lessons in English.

- | | |
|--|------------------------------------|
| 5. I have <i>laid</i> down my pen. | 10. I <i>sat</i> down and rested. |
| 6. The dog has <i>lain</i> down. | 11. I have <i>set</i> it down. |
| 7. <i>Set</i> the pail down. | 12. I have <i>sat</i> down. |
| 8. <i>Sit</i> down and rest. | 13. My work was <i>laid</i> aside. |
| 9. I then <i>set</i> it down. | 14. I was <i>lying</i> down. |
| 15. The trap was <i>set</i> by the river. | |
| 16. I was <i>sitting</i> by the river. | |
| 17. The garment <i>sits</i> well. | |
| 18. The hen <i>sits</i> on her eggs. | |
| 19. He came in and <i>lay</i> down. | |
| 20. The Mediterranean <i>lies</i> between Europe and Africa. | |

We may speak of *laying* something or *setting* something, or may say that something is *laid* or is *set*; but we cannot speak of *lying* or *sitting* something, or of something being *lain* or *sat*. *Set*, in some of its meanings, is used without an object; as, "The sun *set*;" "He *set* out on a journey." *Set* is generally transitive; *sit*, always intransitive. *Lay* is transitive; *lie*, intransitive.

Lay, the present of the first verb, and *lay*, the past of *lie*, may easily be distinguished by the difference in meaning and in the time expressed.

Correct the following errors : —

Those things *have* not *came* to-day.

Wrong; because the past *came* is here used for the past participle *come*. The present perfect tense is formed by prefixing *have* to the past participle.

I done all my work before breakfast.

I come in a little late yesterday.

He has went to my desk without permission

Digitized by Google

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"

Errors in the Copy of the Text

That stupid fellow set down in the room
He sat the chair in the corner
Sit that plate on the table and set it down
I have set in this position a long time
That child will not stay still in the room
I laid down under the tree and set the table
Lie that stick in the table and set it down
Those boys were not in the room
I have made through the room
I done what I could
He has not been in the room
The leaves have not been in the room
This sentence is wrong
He has not been in the room
He has not been in the room
I seen him when he was in the room
My hat was not in the room
The hat was not in the room
I was not in the room
I have been in the room
My head was not in the room
Your head was not in the room
That is not in the room
He has not been in the room
William has not been in the room
You are not in the room
I begin to see what is in the room
The room is not in the room
I thought you were not in the room
The room is not in the room
The room is not in the room

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

264

Graded Lessons In English.

agree in person with the subjects *thou* and *he*; adds *s* in the third sentence and omits it in the first and the fourth to make the verb agree in number with its subjects *he*, *I*, and *they*.

DEFINITION. — **Number** and **person** of a verb are those modifications that show its agreement with the number and person of its subject.

The **infinitive** and the **participle** are forms of the verb that do not assert.

The infinitive is ordinarily found with the preposition *to*, and forms with it the **infinitive phrase** — used as an **adjective**, an **adverb**, or a **noun**.

The participle has an adjective or a noun¹ nature plus its constant verb nature.

DEFINITIONS.

The **infinitive** is a form of the verb which names the action or being in a general way, without asserting it of anything.

The **participle** is a form of the verb partaking of the nature of an adjective or of a noun,¹ and expressing the action or being as assumed.

The **present participle** denotes action or being as continuing at the time indicated by the predicate.

The **past participle** denotes action or being as past or completed at the time indicated by the predicate.

¹ When it has a noun nature it may be called a **nounal verb**. See Lesson 98.

The **past perfect participle** denotes action or being as completed at a time previous to that indicated by the predicate.

Conjugation is the regular arrangement of all the forms of the verb.

Synopsis is the regular arrangement of the forms of one number and person in all the modes and tenses.

The **principal parts** of a verb are the present indicative or infinitive, the past indicative, and the past participle.

Auxiliary verbs are those that help in the conjugation of other verbs.

The auxiliaries are *do, be, have, shall, will, may, can, and must*.

Defective verbs are those verbs some of whose parts are wanting — *can, may, must, ought, shall, and will*.

For the Review of the **conjugation** of verbs — **simple** form, **emphatic** form, **progressive** form, and **passive** form — see Lessons 92, 93, and 94.

THE PREPOSITION.

DEFINITION. — A **preposition** is a word that introduces a phrase modifier, and shows the relation, in sense, of its principal word to the word modified.

Such phrase modifiers have the force of adjectives or adverbs. Care is needed in choosing the right preposition, and in placing the prepositional phrase where it belongs.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

266

Graded Lessons in English.

THE CONJUNCTION.

DEFINITION. — A **conjunction** is a word used to connect words, phrases, or clauses.

Relative pronouns and conjunctive adverbs and prepositions also connect, but the conjunction is the only part of speech which simply connects.

DEFINITIONS.

Coördinate conjunctions are such as connect words, phrases, or clauses of the same rank.

Subordinate conjunctions are such as connect clauses of different rank.

THE INTERJECTION.

DEFINITION. — An **interjection** is a word used to express strong or sudden feeling.

Interjections are without grammatical relation to any word in the sentence.

For a Review of the **paragraph**, of paragraphs forming a **theme**; of **general topic**, **sub-topic**, **framework**, **matter**, and **style**; and of **descriptive**, **narrative**, and **persuasive** writing, see Lessons 30, 40, 50, 60, 70, and 77.

For a Summary of the **rules for capital letters** and **punctuation**, and for illustrative examples, see pp. 224–229.

For **letter-writing** under **heading**, **introduction**, **body of the letter**, **conclusion**, **folding**, and **superscription**, see pp. 229–243.

For a Summary of the **rules of syntax**, see p. 244.

For the ordinary **proof-marks**, see pp. 245, 246.

For **schemes for review**, see pp. 267–270.

For **abbreviations**, see pp. 272–277.

Digitized by Google

SCHEMES FOR REVIEW.

These Schemes will be found very helpful in a general review. The pupils should be able to reproduce them, omitting the Lesson numbers.

SCHEME FOR THE SENTENCE.

(The numbers refer to Lessons.)

PARTS.	Subject.	{	Noun or Pronoun (6, 14, 19).
			Phrase (49).
			Clause (61).
	Predicate.	{	Verb (6, 16).
	Complements.	{	Noun or Pronoun (39).
			Phrase (49).
			Clause (61).
		{	Adjective (39).
			Noun or Pronoun (42).
	Modifiers.	{	Phrase (49).
			Clause (61).
			Adjectives (20, 22).
			Adverbs (24, 27).
			Participles (48).
PARTS.	Connectives.	{	Nouns and Pronouns (53).
			Phrases (31, 48, 49).
			Clauses (57, 59).
			Conjunctions (35, 36, 62).
			Pronouns (57).
PARTS.	Independent Parts	{	Adverbs (59).
			(36, 64).

Classes — Meaning. — Declarative, Interrogative, Imperative, Exclamatory (63).

Classes — Form. — Simple, Complex, Compound (57, 62).

SCHEME FOR THE NOUN.

(The numbers refer to Lessons.)

NOUN (14).	Uses.	{ Subject (6). Object Complement (39). Objective Complement (82). Attribute Complement (42). Adjective Modifier (53). Prin. word in Prep. Phrase (34). Independent (64).	
		{ Common (71). Proper (71).	
	Classes.	Number.	{ Singular (78, 79). Plural (78, 79).
			{ Masculine Feminine } (80). Neuter
	Modifications.	Person.	{ First Second } (81-83). Third
			{ Nominative Possessive } (81-85). Objective

SCHEME FOR THE PRONOUN.

PRONOUNS.	Uses. — Same as those of the Noun.	
	Classes.	{ Personal Relative Interrogative } (71, 72). Adjective
		Modifications. — Same as those of the Noun.

SCHEME FOR THE VERB.

(The numbers refer to Lessons.)

VERB.	Uses.	{ To assert action, being, or state—Predicate (6, 16).	
		{ To assume action, being, or state. { Participles (48). Infinitives (49).	
	Classes.	Form.	{ Regular (74). Irregular (74, 91).
			{ Transitive (74). Intransitive (74).
		Voice.	{ Active (89). Passive (89).
			{ Indicative Potential Subjunctive } (90-94). Imperative
	Modifications.	Tense.	{ Present Past Future Present Perfect Past Perfect Future Perfect } (90-94).
			{ Singular } (90, 92-95). Plural
			{ First Second } (90, 92-95). Third
	Participles. —	Classes.	{ Present Past Past Perfect } (90-94, 96, 98).
	Infinitives. —	Tenses.	{ Present Present Perfect } (90, 92-94).

SCHEME FOR THE ADJECTIVE.

(The numbers refer to Lessons.)

ADJECTIVE.	Uses.	{ Modifier (20, 22). Attribute Complement (39).
	Classes.	{ Descriptive (73). Definitive (73).
	Modification. — Comparison.	{ Pos. Deg. Comp. " (87, 88). Sup. " }

SCHEME FOR THE ADVERB.

ADVERB.	Classes.	{ Time Place Degree Manner } (75).
	Modification. — Comparison.	{ Pos. Deg. Comp. " (87, 88). Sup. " }

SCHEME FOR THE CONJ., PREP., AND INT.

THE CONJUNCTION. — **Classes.** { Coördinate } (36, 76). No
Subordinate } Modifications.

THE PREPOSITION (34, 41). — No **Classes.** No Modifications.

THE INTERJECTION (36). — No **Classes.** No Modifications.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Review of Graded Lessons.

271

Model for Written Parsing adapted to all Parts of Speech. — *Oh ! it has a voice for those who on their sick beds lie and waste away.*

CLASSIFICATION.		MODIFICATIONS.						SYNTAX.			
Sentence.	Class.	Sub-class.	Voice.	Mode.	Tense.	Per.	Num.	Gen.	Case.	Deg. of Comp.	
Oh !	Int.	Per.				3d.	Sing.	Neut.	Nom.		Independent.
it	Pro.	Ir., Tr.				"	"	"			Sub. of <i>has</i> .
has	Vb.	Def.	Act.	Ind.	Pres.	"	"	"			Pred. of <i>it</i> .
a	Adj.	Com.				"	"	"	Obj.		Mod. of <i>voice</i> .
voice	N.										Obj. Com. of <i>has</i> .
for	Prep.										Shows relation of <i>has</i> to <i>those</i> .
those	Pro.	Adj.				"	Plu.	M. or F.	"		Prin. word after <i>for</i> .
who	Pro.	Rel.				"	"	"	Nom.		Sub. of <i>lie</i> and <i>waste</i> .
on	Prep.										Shows relation of <i>lie</i> to <i>beds</i> .
their	Pro.	Per.				"	"	"	Pos.		Poss. Mod. of <i>beds</i> .
sick	Adj.	Des.				"	"	"			Mod. of <i>beds</i> .
beds	N.	Com.				"	"	Neut.	Obj.		Prin. word after <i>on</i> .
lie	Vb.	Ir., Int.				"	"	"			Pred. of <i>who</i> .
and	Conj.	Coör.				"	"	"			Con. <i>lie</i> and <i>waste</i> .
waste	Vb.	Reg., Int.				"	"	"			Pred. of <i>who</i> .
away	Adv.	Place									Mod. of <i>waste</i> .

For exercises in general parsing, select from the preceding Lessons on Analysis.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

272

Graded Lessons in English.

ABBREVIATIONS.

Remarks. — Few abbreviations are allowable in ordinary composition. They are very convenient in writing lists of articles, in scientific works, and wherever certain terms occur frequently.

Titles prefixed to proper names are generally abbreviated, except in addressing an officer of high rank. Titles that immediately follow names are almost always abbreviated.

Names of women are not generally abbreviated, except by using an initial for one of two Christian names.

Abbreviations that shorten only by one letter are unnecessary; as, *Jul.* for "July," *Jno.* for "John," *da.* for "day," etc.

1st, *2d*, *3d*, *4th*, etc. are not followed by the period. They are not treated as abbreviations.

@ , At.	Anon. , Anonymous.
A.B. or B.A. (<i>Artium Baccalaureus</i>), Bachelor of Arts.	Ans. , Answer.
Acct. , acct. , or % , Account.	Anth. , Anthony.
A.D. , (<i>Anno Domini</i>), In the year of our Lord.	Apr. , April.
Adj. , Adjutant.	Arch. , Archibald.
Æt. or æt. (<i>ætatis</i>), Of age, aged.	Ark. , Arkansas.
Ala. , Alabama.	Ariz. , Arizona.
Alex. , Alexander.	Atty. , Attorney.
A.M. or M.A. (<i>Artium Magister</i>), Master of Arts.	Atty-Gen. , Attorney-General.
A.M. or a.m. (<i>ante meridiem</i>), Before noon.	Aug. , August; Augustus.
Amt. , Amount.	Av. or Ave. , Avenue.
And. , Andrew.	Avoir. , Avoirdupois.
	Bart. , Baronet.
	bbl. , Barrels.
	B.C. , Before Christ.
	Benj. , Benjamin.
	Brig-Gen. , Brigadier-General.

Digitized by Google

Abbreviations.

273

B.S. , Bachelor of Science.	Deut. , Deuteronomy.
bu. , Bushel.	D.G. (<i>Dei gratia</i>), By the grace of God.
¢ or ct. , Cents.	Dist.-Atty. , District-Attorney.
Cal. , California.	D.M. , Doctor of Music.
Cap. , Capital. Caps. , Capitals.	do. (<i>ditto</i>), The same.
Capt. , Captain.	doz. , Dozen.
C.E. , Civil Engineer.	Dr. , Doctor; Debtor.
cf. (<i>confer</i>), Compare.	D.V. (<i>Deo volente</i>), God will-
Chas. , Charles.	ing.
Chron. , Chronicles.	E. , East.
Co. , Company; County.	Eben. , Ebenezer.
°/o , In care of.	Ecc. , Ecclesiastes.
C.O.D. , Collect on delivery.	Ed. , Edition; Editor.
Col. , Colonel; Colossians.	Edm. , Edmund.
Coll. , College; Collector.	Edw. , Edward.
Conn. , Connecticut.	e.g. (<i>exempli gratia</i>), For exam-
Cor. , Corinthians.	ple.
Cr. , Credit; Creditor.	E.N.E. , East-northeast.
cub. ft. , Cubic feet.	Eng. , English; England.
cub. in. , Cubic inches.	Eph. , Ephesians; Ephraim.
cwt. , Hundredweight.	E.S.E. , East-southeast.
d. , Days; Pence.	Esq. , Esquire.
Danl. or Dan. , Daniel.	et al. (<i>et alibi</i>), And elsewhere.
D.C. , District of Columbia.	et al. (<i>et alii</i>), And others.
D.C.L. , Doctor of Civil Law.	et seq. (<i>et sequentia</i>), And follow-
D.D. (<i>Divinitatis Doctor</i>), Doc-	ing.
tor of Divinity.	etc. or &c. (<i>et cætera</i>), And
D.D.S. , Doctor of Dental Sur-	others; And so forth.
gery.	Ex. , Example; Exodus.
Dec. , December.	Ez. , Ezra.
Del. , Delaware.	

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

274

Graded Lessons In English.

Ezek. , Ezekiel.	H.R.H. , His (or Her) Royal Highness.
Fahr. or F. , Fahrenheit (thermometer).	ib. or ibid (<i>ibidem</i>), In the same place.
Feb. , February.	id. (<i>idem</i>), The same.
Fla. , Florida.	i.e. (<i>id est</i>), That is.
Fr. , French ; France.	I.H.S. (<i>Jesus hominum Salvator</i>), Jesus the Savior of Men.
Fran. , Francis.	Ill. , Illinois.
Fred. , Frederic.	in. , Inches.
Fri. , Friday.	incog. (<i>incognito</i>), Unknown.
ft. , Feet.	Ind. , Indiana.
Ft. , Fort.	Ind. T. , Indian Territory.
fur. , Furlong.	inst. , Instant, the present month.
Ga. , Georgia.	Io. , Iowa.
Gal. , Galatians.	I.O.O.F. , Independent Order of Odd Fellows.
gal. , Gallons.	Isa. , Isaiah.
Gen. , General ; Genesis.	Jac. , Jacob.
Geo. , George.	Jan. , January.
Gov. , Governor.	Jas. , James.
gr. , Grains.	Jer. , Jeremiah.
h. , Hours.	Jona. , Jonathan.
Hab. , Habakkuk.	Jos. , Joseph.
Hag. , Haggai.	Josh. , Joshua.
H.B.M. , His (or Her) Britannic Majesty.	Jr. or Jun. , Junior.
hdkf. , Handkerchief.	Judg. , Judges.
Heb. , Hebrews.	Kans. or Kan. , Kansas.
H.H. , His Holiness (the Pope).	Ky. , Kentucky.
hhd. , Hogsheads.	L. , Latin.
H.M. , His (or Her) Majesty.	L. , Line ; ll. , Lines.
Hon. , Honorable.	
Hos. , Hosea.	

Digitized by Google

l. or £ , Pounds sterling.	Mic. , Micah.
La. , Louisiana.	Mich. , Michigan ; Michael.
Lam. , Lamentations.	Minn. , Minnesota.
lb. or lb. (<i>libra</i> or <i>libræ</i>), Pound or pounds in weight.	Miss. , Mississippi.
l.c. , Lower case (small letter).	Mlle. , Mademoiselle.
Lev. , Leviticus.	Mmes. , Mesdames.
L.I. , Long Island.	Mo. , Missouri.
Lieut. , Lieutenant.	mo. , Months.
LL.B. (<i>Legum Baccalaureus</i>), Bachelor of Laws.	Mon. , Monday.
LL.D. (<i>Legum Doctor</i>), Doctor of Laws.	M.P. , Member of Parliament.
M. or Mons. , Monsieur.	Mont. , Montana.
M. (<i>meridies</i>), Noon.	Mr. , Mister.
m. , Miles ; Minutes.	Mrs. , Mistress (pronounced Missis).
Mad. , Madam. Mme. , Madame.	MS. , Manuscript.
Maj. , Major.	MSS. , Manuscripts.
Mal. , Malachi.	Mt. , Mountain.
Mar. , March.	N. , North.
Mass. , Massachusetts.	N.A. , North America.
Matt. , Matthew.	Nath. , Nathaniel.
M.C. , Member of Congress.	N.B. (<i>nota bene</i>), Mark well.
M.D. (<i>Medicine Doctor</i>), Doctor of Medicine.	N.C. , North Carolina.
Md. , Maryland.	N.Dak. , North Dakota.
mdse. , Merchandise.	N.E. , New England.
Me. , Maine.	N.E. , Northeast.
Mem. , Memorandum ; Memoranda.	Neh. , Nehemiah.
Messrs. , Messieurs.	Neth. , Netherlands.
Mgr. , Monseigneur.	Nev. , Nevada.
	N.H. , New Hampshire.
	N.J. , New Jersey.
	N.Mex. or N.M. , New Mexico.
	N.N.E. , North-northeast.

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

276

Graded Lessons in English.

N.N.W. , North-northwest.	prox. (<i>proximo</i>), The next month.
N.O. , New Orleans.	P.S. , Postscript.
No. (<i>numero</i>), Number.	Ps. , Psalms.
Nov. , November.	pt. , Pints.
N.W. , Northwest.	pwt. , Pennyweights.
N.Y. , New York.	qt. , Quarts.
O. , Ohio.	q.v. (<i>quod vide</i>), Which see.
Obad. , Obadiah.	Qy. , Query.
Oct. , October.	rd. , Rods.
Oreg. or Or. , Oregon.	Recd. , Received.
Oxon. (<i>Oxonia</i>), Oxford.	Rev. , Reverend ; Revelation.
oz. , Ounces.	R.I. , Rhode Island.
p. , Page. pp. , Pages.	Robt. , Robert.
Pa. or Penn. , Pennsylvania.	Rom. , Romans (Book of) ; Roman letters.
Payt. or payt. , Payment.	R.R. , Railroad.
per cent or per ct. or % (<i>per centum</i>), By the hundred.	R.S.V.P. (<i>Répondez s'il vous plait</i>), Answer, if you please.
Ph.D. (<i>Philosophiæ Doctor</i>), Doctor of Philosophy.	Rt. Hon. , Right Honorable.
Phil. , Philip ; Philippians.	Rt. Rev. , Right Reverend.
Phila. , Philadelphia.	S. , South.
pk. , Pecks.	s. , Shillings.
P.M. , Postmaster.	S.A. , South America.
P.M. or p.m. (<i>post meridiem</i>), Afternoon.	Saml. or Sam. , Samuel.
P.O. , Post Office.	Sat. , Saturday.
Pres. , President.	S.C. , South Carolina.
Prof. , Professor.	S. Dak. , South Dakota.
Pro tem. (<i>pro tempore</i>), For the time being.	S.E. , Southeast.
Prov. , Proverbs.	Sec. , Secretary.
	sec. , seconds.

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Abbreviations.

277

Sep. or Sept. , September.	of America; United States Army.
Sol. , Solomon.	U.S.M. , United States Mail.
sq. ft. , Square feet.	U.S.N. , United States Navy.
sq. in. , Square inches.	Va. , Virginia.
sq. m. , Square miles.	Vice-Pres. , Vice-President.
S.S.E. , South-southeast.	viz. (videlicet) , To wit, namely.
S.S.W. , South-southwest.	vol. , Volume.
St. , Street; Saint.	vs. (versus) , Against.
Sun. , Sunday.	Vt. , Vermont.
Supt. , Superintendent.	W. , West.
S.W. , Southwest.	Wed. , Wednesday.
T. , Tons; Tuns.	w.f. , Wrong font.
Tenn. , Tennessee.	Wis. , Wisconsin.
Tex. , Texas.	wk. , Weeks.
Theo. , Theodore.	Wm. , William.
Theoph. , Theophilus.	W.N.W. , West-northwest.
Thess. , Thessalonians.	W.S.W. , West-southwest.
Thos. , Thomas.	W. Va. , West Virginia.
Thurs. , Thursday.	Wyo. , Wyoming.
Tim. , Timothy.	Xmas , Christmas.
tr. , Transpose.	yd. , Yards.
Treas. , Treasurer.	y. or yr. , Years.
Tues. , Tuesday.	Zech. , Zechariah.
ult. (ultimo) , Last — last month.	& Co. , And Company.
U.S. or U.S.A. , United States	

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Digitized by Google

Volume Orange Issue Four "Over the Horizon"

		PAGE			PAGE
Abbreviations		77, 78, 272-277	Colon		189, 227
	abuse of	43, 44	Comma		76, 77, 118, 127, 226
	<i>an</i> and <i>a</i>	42	Complements	attribute	80, 81, 95, 112
	arrangement of	52, 53		object	80, 81
	choice of	41-44, 54, 55		objective	183, 184
Adjectives			Composition		18
	classes of	161, 162	Conjugation		201
	comparison of	192-196	Conjunctions	classes of	167
	definition of	39, 161		definition of	72, 168
	distinguished from adverbs	83, 84	Connectives	list of	169, 170
	effect upon style	41	Contraction		148-151
	"scheme" for review	270	Diagrams	definition	17
	arrangement of	82-84, 113		use of	5, 6
	classes of	166	Expansion		148-151
	comparison of	194-196	Exposition		145, 146
	conjective	167, 169, 170	Figurative Expressions		154
Adverbs			Framework		89, 118, 124
	definition of	49	Grammar, English		12
	distinguished from adjectives		Indirect Object		185
		88, 84	Infinitive Phrase		105-107
	"scheme" for review	270	Interjections		72, 73
	use of	43-52	Interrogation Point		142, 225
Agreement		18, 22, 74-76	Knowledge	first hand	59, 60
Analysis, oral		26, 81, 35, 45, 47, 49, 68, 68, 71, 73, 82, 104, 106, 116, 126, 180, 187, 140		second hand	59, 60
Antecedent		160	Language	natural	12
Argument		128		talk on	9-14
Arrangement		90-94		verbal	12
Articles		38, 42	Letters	consonants	11, 12
Auxiliary Verbs		201		sonants	11, 12
Be		209-212		surds	11, 12
Capital Letters, rules		19, 27, 81, 77, 162, 224, 225		vowels	11, 12
	arrangement of	180-182		what	9-13
	definition of	125	Letter-Writing		229-243
Classes			Modifications		176, 177
	dependent				
	adjective	124-128			
	adverb	129-132			
	noun	136-139			
	independent	125, 140, 141			

279

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

280

Graded Lessons in English.

	PAGE		PAGE
Modifier,	84	Participle	102-105
adjective.....	88-44	Parts of Speech	25
adverb.....	44-52	Period	19, 77, 225
clause.....	118, 119, 194, 182	arrangement.....	76, 77, 90-94
noun.....	115, 116, 118	change of.....	66, 67
phrase.....	61-69, 118, 119	complex.....	101
Modified Predicate	44, 45	compound.....	101
Modified Subject	88, 84	definition.....	62
Negatives	94	Phrases,	
Nominative Forms	190	discussion of.....	60-62
Nounal Verb	219 (note)	infinitive.....	105-107
classes of.....	157, 158	participial.....	107, 108
declension of.....	185	position of.....	90, 99
definition of.....	25, 26	prepositional.....	60-64
Nouns,		transposed.....	90, 91
case.....	182, 185, 188-190	Possessive Forms	188-190
gender.....	178, 179	Predicate	15-24, 27-29, 86, 44, 45, 70, 79, 81
number.....	74, 75, 175-178	Prepositions	67, 68, 94
person.....	180-182	agreement of.....	76, 191
modifiers, as.....	115, 116, 185	classes of.....	158, 159, 187, 188
"scheme" for review.....	269	declension of.....	186-188
ten offices of.....	251	definition of.....	81, 159
Object	164 (note)	discussion of.....	80-82
Objective Forms	190	Pronouns,	
Objects writers have	158, 154	case.....	180-185, 190
Order, usual and transposed	69, 90-94	modification of.....	
composition of.....	55-60, 87, 89, 108-118, 155, 156, 178, 174	gender.....	178, 179
definition of.....	55, 56, 58, 156	number.....	187-189
framework for.....	89, 118, 124	person.....	180-185
descriptive.....	108-118, 171-173	modifiers, as.....	115-117, 184
explanatory.....	56-60	"scheme" for review.....	268
kinds of, illustrated,		Proof-Marks	245, 246
narrative.....	85-89, 188-185	colon.....	189, 227
persuasive.....	188-185, 152-154	comma.....	76, 77, 118, 127, 226
length of.....	124	exclamation point.....	78, 225
material for.....	59, 60	(explanation of restrictive).....	118, 119
order of.....	89, 178	Punctuation, Rules for,	
relation of.....	58, 59, 68, 89	interrogation point.....	142, 225
topics and sub-topics of.....	89, 124	period.....	19, 77, 225
Parsing	26, 29, 81, 89, 78, 106, 180, 192, 218, 271	quotation marks.....	185, 188, 189
		semicolon.....	184, 188, 227
		summary of rules for all points.....	225-229
		Quotations,	
		direct.....	188, 189
		indirect.....	188, 189

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Index.

281

	PAGE		PAGE
	258, 259	Syntax, rules of	244
	259, 260	Synthesis	18
	266	Theme	124
Review of Graded Lessons,	266	To, with infinitive	105, 106, 118
	251-255	agreement of	17-20, 22, 82, 86, 74-76, 218-215
	265		162-164
	256-258	classes,	164, 165, 201, 202
	247-251		163-165
	260-265		164 (note)
Review Questions	15, 21, 27, 86, 87, 47, 48, 55, 71, 84, 108, 182, 188, 190-192, 170, 171, 179, 180, 191, 195, 196, 218		162-164 (note)
See	208-205		163-165
S-Ending	18-20, 22, 82, 86, 74, 75, 120		162-164
	16, 17	classified	163-165
	14		162-164
	16, 17	conjugation of	201-212
Sentences,	15	definition of	28
	16, 21, 29, 45, 70, 79-82	emphatic form	206
	8-5	infinitive, the	105, 200
	16, 17, 26, 70		198-200
	126-182	number	199, 200
	140, 141	modifications of	199, 200
	126		198, 200
Sentences (classes),	141, 142	voice	196, 197
	141, 142	participle	102-105, 200, 219 (note), 220
	141, 142		215-218
	141, 142	passive form	212
	24, 141, 142	principal parts	201
Sounds	9-12	progressive form	212
Subject	15-24, 26, 82, 85, 70, 106, 107, 186, 187	"scheme" for review	260
Synopsis	201	What	160

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Digitized by Google✓

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"

This book should be returned to
the Library on or before the last date
stamped below.

A fine is incurred by retaining it
beyond the specified time.

Please return promptly.

15 5 6 7 8
CANCELLED

Digitized by Google

The Uncertainty Principle

Volume Orange Issue Four "Over the Horizon"



The Adventures of Tom Sawyer

Contents

Articles

The Adventures of Tom Sawyer	1
The Adventures of Tom Sawyer/Chapter I	2
The Adventures of Tom Sawyer/Chapter II	6
The Adventures of Tom Sawyer/Chapter III	9
The Adventures of Tom Sawyer/Chapter IV	12
The Adventures of Tom Sawyer/Chapter V	17
The Adventures of Tom Sawyer/Chapter VI	19
The Adventures of Tom Sawyer/Chapter VII	25
The Adventures of Tom Sawyer/Chapter VIII	29
The Adventures of Tom Sawyer/Chapter IX	31
The Adventures of Tom Sawyer/Chapter X	35
The Adventures of Tom Sawyer/Chapter XI	38
The Adventures of Tom Sawyer/Chapter XII	41
The Adventures of Tom Sawyer/Chapter XIII	43
The Adventures of Tom Sawyer/Chapter XIV	47
The Adventures of Tom Sawyer/Chapter XV	50
The Adventures of Tom Sawyer/Chapter XVI	52
The Adventures of Tom Sawyer/Chapter XVII	56
The Adventures of Tom Sawyer/Chapter XVIII	58
The Adventures of Tom Sawyer/Chapter XIX	62
The Adventures of Tom Sawyer/Chapter XX	63
The Adventures of Tom Sawyer/Chapter XXI	66
The Adventures of Tom Sawyer/Chapter XXII	69
The Adventures of Tom Sawyer/Chapter XXIII	70
The Adventures of Tom Sawyer/Chapter XXIV	74
The Adventures of Tom Sawyer/Chapter XXV	75
The Adventures of Tom Sawyer/Chapter XXVI	79
The Adventures of Tom Sawyer/Chapter XXVII	83
The Adventures of Tom Sawyer/Chapter XXVIII	85
The Adventures of Tom Sawyer/Chapter XXIX	87
The Adventures of Tom Sawyer/Chapter XXX	90
The Adventures of Tom Sawyer/Chapter XXXI	95
The Adventures of Tom Sawyer/Chapter XXXII	99
The Adventures of Tom Sawyer/Chapter XXXIII	100

The Adventures of Tom Sawyer/Chapter XXXIV	105
The Adventures of Tom Sawyer/Chapter XXXV	107
The Adventures of Tom Sawyer/Conclusion	109

References

Article Sources and Contributors	110
Image Sources, Licenses and Contributors	111

Article Licenses

License	112
---------	-----

The Adventures of Tom Sawyer

PREFACE

Most of the adventures recorded in this book really occurred; one or two were experiences of my own, the rest those of boys who were schoolmates of mine. Huck Finn is drawn from life; Tom Sawyer also, but not from an individual—he is a combination of the characteristics of three boys whom I knew, and therefore belongs to the composite order of architecture. The odd superstitions touched upon were all prevalent among children and slaves in the West at the period of this story—that is to say, thirty or forty years ago.

Although my book is intended mainly for the entertainment of boys and girls, I hope it will not be shunned by men and women on that account, for part of my plan has been to try to pleasantly remind adults of what they once were themselves, and of how they felt and thought and talked, and what queer enterprises they sometimes engaged in.

The Author. Hartford, 1876.

- Chapter I
- Chapter II
- Chapter III
- Chapter IV
- Chapter V
- Chapter VI
- Chapter VII
- Chapter VIII
- Chapter IX
- Chapter X
- Chapter XI
- Chapter XII
- Chapter XIII
- Chapter XIV
- Chapter XV
- Chapter XVI
- Chapter XVII
- Chapter XVIII
- Chapter XIX
- Chapter XX
- Chapter XXI
- Chapter XXII
- Chapter XXIII
- Chapter XXIV
- Chapter XXV
- Chapter XXVI 🎧
- Chapter XXVII
- Chapter XXVIII
- Chapter XXIX 🎧
- Chapter XXX 🎧
- Chapter XXXI
- Chapter XXXII
- Chapter XXXIII

- Chapter XXXIV
- Chapter XXXV
- Conclusion



This work published before January 1, 1923 is in the **public domain** worldwide because the author died at least 100 years ago.

The Adventures of Tom Sawyer/Chapter I

"Tom!"

No answer.

"Tom!"

No answer.

"What's gone with that boy, I wonder? You Tom!"

No answer.

The old lady pulled her spectacles down and looked over them about the room; then she put them up and looked out under them. She seldom or never looked through them for so small a thing as a boy; they were her state pair, the pride of her heart, and were built for "style," not service -- she could have seen through a pair of stove-lids just as well. She looked perplexed for a moment, and then said, not fiercely, but still loud enough for the furniture to hear:

"Well, I lay if I get hold of you I'll --"

She did not finish, for by this time she was bending down and punching under the bed with the broom, and so she needed breath to punctuate the punches with. She resurrected nothing but the cat.

"I never did see the beat of that boy!"

She went to the open door and stood in it and looked out among the tomato vines and "jimpson" weeds that constituted the garden. No Tom. So she lifted up her voice at an angle calculated for distance and shouted:

"Y-o-u-u Tom!"

There was a slight noise behind her and she turned just in time to seize a small boy by the slack of his roundabout and arrest his flight.

"There! I might 'a' thought of that closet. What you been doing in there?"

"Nothing."

"Nothing! Look at your hands. And look at your mouth. What is that?"

"I don't know, aunt."

"Well, I know. It's jam -- that's what it is. Forty times I've said if you didn't let that jam alone I'd skin you. Hand me that switch."

The switch hovered in the air -- the peril was desperate --

"My! Look behind you, aunt!"

The old lady whirled round, and snatched her skirts out of danger. The lad fled on the instant, scrambled up the high board-fence, and disappeared over it.

His aunt Polly stood surprised a moment, and then broke into a gentle laugh.

"Hang the boy, can't I never learn anything? Ain't he played me tricks enough like that for me to be looking out for him by this time? But old fools is the biggest fools there is. Can't learn an old dog new tricks, as the saying is. But my goodness, he never plays them alike, two days, and how is a body to know what's coming? He 'pears to know just

how long he can torment me before I get my dander up, and he knows if he can make out to put me off for a minute or make me laugh, it's all down again and I can't hit him a lick. I ain't doing my duty by that boy, and that's the Lord's truth, goodness knows. Spare the rod and spoil the child, as the Good Book says. I'm a laying up sin and suffering for us both, I know. He's full of the Old Scratch, but laws-a-me! he's my own dead sister's boy, poor thing, and I ain't got the heart to lash him, somehow. Every time I let him off, my conscience does hurt me so, and every time I hit him my old heart most breaks. Well-a-well, man that is born of woman is of few days and full of trouble, as the Scripture says, and I reckon it's so. He'll play hookey this evening [*], and I'll just be obliged to make him work, to-morrow, to punish him. It's mighty hard to make him work Saturdays, when all the boys is having holiday, but he hates work more than he hates anything else, and I've got to do some of my duty by him, or I'll be the ruination of the child."

Tom did play hookey, and he had a very good time. He got back home barely in season to help Jim, the small colored boy, saw next-day's wood and split the kindlings before supper -- at least he was there in time to tell his adventures to Jim while Jim did three-fourths of the work. Tom's younger brother (or rather half-brother) Sid was already through with his part of the work (picking up chips), for he was a quiet boy, and had no adventurous, troublesome ways.

While Tom was eating his supper, and stealing sugar as opportunity offered, Aunt Polly asked him questions that were full of guile, and very deep -- for she wanted to trap him into damaging revealments. Like many other simple-hearted souls, it was her pet vanity to believe she was endowed with a talent for dark and mysterious diplomacy, and she loved to contemplate her most transparent devices as marvels of low cunning. Said she:

"Tom, it was middling warm in school, warn't it?"

"Yes'm."

"Powerful warm, warn't it?"

"Yes'm."

"Didn't you want to go in a-swimming, Tom?"

A bit of a scare shot through Tom -- a touch of uncomfortable suspicion. He searched Aunt Polly's face, but it told him nothing. So he said:

"No'm -- well, not very much."

The old lady reached out her hand and felt Tom's shirt, and said:

"But you ain't too warm now, though." And it flattered her to reflect that she had discovered that the shirt was dry without anybody knowing that that was what she had in her mind. But in spite of her, Tom knew where the wind lay, now. So he forestalled what might be the next move:

"Some of us pumped on our heads -- mine's damp yet. See?"

Aunt Polly was vexed to think she had overlooked that bit of circumstantial evidence, and missed a trick. Then she had a new inspiration:

"Tom, you didn't have to undo your shirt collar where I sewed it, to pump on your head, did you? Unbutton your jacket!"

The trouble vanished out of Tom's face. He opened his jacket. His shirt collar was securely sewed.

"Bother! Well, go 'long with you. I'd made sure you'd played hookey and been a-swimming. But I forgive ye, Tom. I reckon you're a kind of a singed cat, as the saying is -- better'n you look. This time."

She was half sorry her sagacity had miscarried, and half glad that Tom had stumbled into obedient conduct for once.

But Sidney said:

"Well, now, if I didn't think you sewed his collar with white thread, but it's black."

"Why, I did sew it with white! Tom!"

But Tom did not wait for the rest. As he went out at the door he said:

"Siddy, I'll lick you for that."

In a safe place Tom examined two large needles which were thrust into the lapels of his jacket, and had thread bound about them -- one needle carried white thread and the other black. He said:

"She'd never noticed if it hadn't been for Sid. Confound it! sometimes she sews it with white, and sometimes she sews it with black. I wish to geeminy she'd stick to one or t'other -- I can't keep the run of 'em. But I bet you I'll lam Sid for that. I'll learn him!"

He was not the Model Boy of the village. He knew the model boy very well though -- and loathed him.

Within two minutes, or even less, he had forgotten all his troubles. Not because his troubles were one whit less heavy and bitter to him than a man's are to a man, but because a new and powerful interest bore them down and drove them out of his mind for the time -- just as men's misfortunes are forgotten in the excitement of new enterprises. This new interest was a valued novelty in whistling, which he had just acquired from a negro, and he was suffering to practise it undisturbed. It consisted in a peculiar bird-like turn, a sort of liquid warble, produced by touching the tongue to the roof of the mouth at short intervals in the midst of the music -- the reader probably remembers how to do it, if he has ever been a boy. Diligence and attention soon gave him the knack of it, and he strode down the street with his mouth full of harmony and his soul full of gratitude. He felt much as an astronomer feels who has discovered a new planet -- no doubt, as far as strong, deep, unalloyed pleasure is concerned, the advantage was with the boy, not the astronomer.

The summer evenings were long. It was not dark, yet. Presently Tom checked his whistle. A stranger was before him -- a boy a shade larger than himself. A new-comer of any age or either sex was an impressive curiosity in the poor little shabby village of St. Petersburg. This boy was well dressed, too -- well dressed on a week-day. This was simply astounding. His cap was a dainty thing, his closebuttoned blue cloth roundabout was new and natty, and so were his pantaloons. He had shoes on -- and it was only Friday. He even wore a necktie, a bright bit of ribbon. He had a citified air about him that ate into Tom's vitals. The more Tom stared at the splendid marvel, the higher he turned up his nose at his finery and the shabbier and shabbier his own outfit seemed to him to grow. Neither boy spoke. If one moved, the other moved -- but only sidewise, in a circle; they kept face to face and eye to eye all the time. Finally Tom said:

"I can lick you!"

"I'd like to see you try it."

"Well, I can do it."

"No you can't, either."

"Yes I can."

"No you can't."

"I can."

"You can't."

"Can!"

"Can't!"

An uncomfortable pause. Then Tom said:

"What's your name?"

"Tisn't any of your business, maybe."

"Well I 'low I'll make it my business."

"Well why don't you?"

"If you say much, I will."

"Much -- much -- much. There now."

"Oh, you think you're mighty smart, don't you? I could lick you with one hand tied behind me, if I wanted to."

"Well why don't you do it? You say you can do it."

"Well I will, if you fool with me."

"Oh yes -- I've seen whole families in the same fix."

"Smarty! You think you're some, now, don't you? Oh, what a hat!"

"You can lump that hat if you don't like it. I dare you to knock it off -- and anybody that'll take a dare will suck eggs."

"You're a liar!"

"You're another."

"You're a fighting liar and dasn't take it up."

"Aw -- take a walk!"

"Say -- if you give me much more of your sass I'll take and bounce a rock off'n your head."

"Oh, of course you will."

"Well I will."

"Well why don't you do it then? What do you keep saying you will for? Why don't you do it? It's because you're afraid."

"I ain't afraid."

"You are."

"I ain't."

"You are."

Another pause, and more eying and sidling around each other. Presently they were shoulder to shoulder. Tom said:

"Get away from here!"

"Go away yourself!"

"I won't."

"I won't either."

So they stood, each with a foot placed at an angle as a brace, and both shoving with might and main, and glowering at each other with hate. But neither could get an advantage. After struggling till both were hot and flushed, each relaxed his strain with watchful caution, and Tom said:

"You're a coward and a pup. I'll tell my big brother on you, and he can thrash you with his little finger, and I'll make him do it, too."

"What do I care for your big brother? I've got a brother that's bigger than he is -- and what's more, he can throw him over that fence, too." [Both brothers were imaginary.]

"That's a lie."

"Your saying so don't make it so."

Tom drew a line in the dust with his big toe, and said:

"I dare you to step over that, and I'll lick you till you can't stand up. Anybody that'll take a dare will steal sheep."

The new boy stepped over promptly, and said:

"Now you said you'd do it, now let's see you do it."

"Don't you crowd me now; you better look out."

"Well, you said you'd do it -- why don't you do it?"

"By jingo! for two cents I will do it."

The new boy took two broad coppers out of his pocket and held them out with derision. Tom struck them to the ground. In an instant both boys were rolling and tumbling in the dirt, gripped together like cats; and for the space of a minute they tugged and tore at each other's hair and clothes, punched and scratched each other's nose, and covered themselves with dust and glory. Presently the confusion took form, and through the fog of battle Tom appeared, seated astride the new boy, and pounding him with his fists. "Holler 'nuff!" said he.

The boy only struggled to free himself. He was crying -- mainly from rage.

"Holler 'nuff!" -- and the pounding went on.

At last the stranger got out a smothered "'Nuff!" and Tom let him up and said:

"Now that'll learn you. Better look out who you're fooling with next time."

The new boy went off brushing the dust from his clothes, sobbing, snuffling, and occasionally looking back and shaking his head and threatening what he would do to Tom the "next time he caught him out." To which Tom responded with jeers, and started off in high feather, and as soon as his back was turned the new boy snatched up a stone, threw it and hit him between the shoulders and then turned tail and ran like an antelope. Tom chased the traitor home, and thus found out where he lived. He then held a position at the gate for some time, daring the enemy to come outside, but the enemy only made faces at him through the window and declined. At last the enemy's mother appeared, and called Tom a bad, vicious, vulgar child, and ordered him away. So he went away; but he said he "'lowed" to "lay" for that boy.

He got home pretty late that night, and when he climbed cautiously in at the window, he uncovered an ambush, in the person of his aunt; and when she saw the state his clothes were in her resolution to turn his Saturday holiday into captivity at hard labor became adamant in its firmness.

The Adventures of Tom Sawyer/Chapter II

Saturday morning was come, and all the summer world was bright and fresh, and brimming with life. There was a song in every heart; and if the heart was young the music issued at the lips. There was cheer in every face and a spring in every step. The locust-trees were in bloom and the fragrance of the blossoms filled the air. Cardiff Hill, beyond the village and above it, was green with vegetation and it lay just far enough away to seem a Delectable Land, dreamy, reposeful, and inviting.

Tom appeared on the sidewalk with a bucket of whitewash and a long-handled brush. He surveyed the fence, and all gladness left him and a deep melancholy settled down upon his spirit. Thirty yards of board fence nine feet high. Life to him seemed hollow, and existence but a burden. Sighing, he dipped his brush and passed it along the topmost plank; repeated the operation; did it again; compared the insignificant whitewashed streak with the far-reaching continent of unwhitewashed fence, and sat down on a tree-box discouraged. Jim came skipping out at the gate with a tin pail, and singing Buffalo Gals. Bringing water from the town pump had always been hateful work in Tom's eyes, before, but now it did not strike him so. He remembered that there was company at the pump. White, mulatto, and negro boys and girls were always there waiting their turns, resting, trading playthings, quarrelling, fighting, skylarking. And he remembered that although the pump was only a hundred and fifty yards off, Jim never got back with a bucket of water under an hour -- and even then somebody generally had to go after him. Tom said:

"Say, Jim, I'll fetch the water if you'll whitewash some."

Jim shook his head and said:

"Can't, Mars Tom. Ole missis, she tole me I got to go an' git dis water an' not stop foolin' roun' wid anybody. She say she spec' Mars Tom gwine to ax me to whitewash, an' so she tole me go 'long an' 'tend to my own business -- she 'lowed she'd 'tend to de whitewashin'."

"Oh, never you mind what she said, Jim. That's the way she always talks. Gimme the bucket -- I won't be gone only a minute. She won't ever know."

"Oh, I dasn't, Mars Tom. Ole missis she'd take an' tar de head off'n me. 'Deed she would."

"She! She never licks anybody -- whacks 'em over the head with her thimble -- and who cares for that, I'd like to know. She talks awful, but talk don't hurt -- anyways it don't if she don't cry. Jim, I'll give you a marvel. I'll give you a white alley!"

Jim began to waver.

"White alley, Jim! And it's a bully taw."

"My! Dat's a mighty gay marvel, I tell you! But Mars Tom I's powerful 'fraid ole missis --"

"And besides, if you will I'll show you my sore toe."

Jim was only human -- this attraction was too much for him. He put down his pail, took the white alley, and bent over the toe with absorbing interest while the bandage was being unwound. In another moment he was flying down the street with his pail and a tingling rear, Tom was whitewashing with vigor, and Aunt Polly was retiring from the field with a slipper in her hand and triumph in her eye.

But Tom's energy did not last. He began to think of the fun he had planned for this day, and his sorrows multiplied. Soon the free boys would come tripping along on all sorts of delicious expeditions, and they would make a world of fun of him for having to work -- the very thought of it burnt him like fire. He got out his worldly wealth and examined it -- bits of toys, marbles, and trash; enough to buy an exchange of work, maybe, but not half enough to buy so much as half an hour of pure freedom. So he returned his straitened means to his pocket, and gave up the idea of trying to buy the boys. At this dark and hopeless moment an inspiration burst upon him! Nothing less than a great, magnificent inspiration.

He took up his brush and went tranquilly to work. Ben Rogers hove in sight presently -- the very boy, of all boys, whose ridicule he had been dreading. Ben's gait was the hop-skip-and-jump -- proof enough that his heart was light and his anticipations high. He was eating an apple, and giving a long, melodious whoop, at intervals, followed by a deep-toned ding-dong-dong, ding-dong-dong, for he was personating a steamboat. As he drew near, he slackened speed, took the middle of the street, leaned far over to starboard and rounded to ponderously and with laborious pomp and circumstance -- for he was personating the Big Missouri, and considered himself to be drawing nine feet of water. He was boat and captain and engine-bells combined, so he had to imagine himself standing on his own hurricane-deck giving the orders and executing them:

"Stop her, sir! Ting-a-ling-ling!" The headway ran almost out, and he drew up slowly toward the sidewalk.

"Ship up to back! Ting-a-ling-ling!" His arms straightened and stiffened down his sides.

"Set her back on the stabboard! Ting-a-ling-ling! Chow! ch-chow-wow! Chow!" His right hand, meantime, describing stately circles -- for it was representing a forty-foot wheel.

"Let her go back on the labboard! Ting-a-ling-ling! Chow-ch-chow-chow!" The left hand began to describe circles.

"Stop the stabboard! Ting-a-ling-ling! Stop the labboard! Come ahead on the stabboard! Stop her! Let your outside turn over slow! Ting-a-ling-ling! Chow-ow-ow! Get out that head-line! Lively now! Come -- out with your spring-line -- what're you about there! Take a turn round that stump with the bight of it! Stand by that stage, now -- let her go! Done with the engines, sir! Ting-a-ling-ling! Sh't! S'h't! Sh't!" (trying the gauge-cocks).

Tom went on whitewashing -- paid no attention to the steamboat. Ben stared a moment and then said: "Hi-yi! You're up a stump, ain't you!"

No answer. Tom surveyed his last touch with the eye of an artist, then he gave his brush another gentle sweep and surveyed the result, as before. Ben ranged up alongside of him. Tom's mouth watered for the apple, but he stuck to his work. Ben said:

"Hello, old chap, you got to work, hey?"

Tom wheeled suddenly and said:

"Why, it's you, Ben! I warn't noticing."

"Say -- I'm going in a-swimming, I am. Don't you wish you could? But of course you'd druther work -- wouldn't you? Course you would!"

Tom contemplated the boy a bit, and said:

"What do you call work?"

"Why, ain't that work?"

Tom resumed his whitewashing, and answered carelessly:

"Well, maybe it is, and maybe it ain't. All I know, is, it suits Tom Sawyer."

"Oh come, now, you don't mean to let on that you like it?"

The brush continued to move.

"Like it? Well, I don't see why I oughtn't to like it. Does a boy get a chance to whitewash a fence every day?"

That put the thing in a new light. Ben stopped nibbling his apple. Tom swept his brush daintily back and forth -- stepped back to note the effect -- added a touch here and there -- criticised the effect again -- Ben watching every move and getting more and more interested, more and more absorbed. Presently he said:

"Say, Tom, let me whitewash a little."

Tom considered, was about to consent; but he altered his mind:

"No -- no -- I reckon it wouldn't hardly do, Ben. You see, Aunt Polly's awful particular about this fence -- right here on the street, you know -- but if it was the back fence I wouldn't mind and she wouldn't. Yes, she's awful particular about this fence; it's got to be done very careful; I reckon there ain't one boy in a thousand, maybe two thousand, that can do it the way it's got to be done."

"No -- is that so? Oh come, now -- lemme just try. Only just a little -- I'd let you, if you was me, Tom."

"Ben, I'd like to, honest injun; but Aunt Polly -- well, Jim wanted to do it, but she wouldn't let him; Sid wanted to do it, and she wouldn't let Sid. Now don't you see how I'm fixed? If you was to tackle this fence and anything was to happen to it --"

"Oh, shucks, I'll be just as careful. Now lemme try. Say -- I'll give you the core of my apple."

"Well, here -- No, Ben, now don't. I'm afeard --"

"I'll give you all of it!"

Tom gave up the brush with reluctance in his face, but alacrity in his heart. And while the late steamer Big Missouri worked and sweated in the sun, the retired artist sat on a barrel in the shade close by, dangled his legs, munched his apple, and planned the slaughter of more innocents. There was no lack of material; boys happened along every little while; they came to jeer, but remained to whitewash. By the time Ben was fagged out, Tom had traded the next chance to Billy Fisher for a kite, in good repair; and when he played out, Johnny Miller bought in for a dead rat and a string to swing it with -- and so on, and so on, hour after hour. And when the middle of the afternoon came, from being a poor poverty-stricken boy in the morning, Tom was literally rolling in wealth. He had besides the things before mentioned, twelve marbles, part of a jews-harp, a piece of blue bottle-glass to look through, a spool cannon, a key that wouldn't unlock anything, a fragment of chalk, a glass stopper of a decanter, a tin soldier, a couple of tadpoles, six fire-crackers, a kitten with only one eye, a brass doorknob, a dog-collar -- but no dog -- the handle of a knife, four pieces of orange-peel, and a dilapidated old window sash.

He had had a nice, good, idle time all the while -- plenty of company -- and the fence had three coats of whitewash on it! If he hadn't run out of whitewash he would have bankrupted every boy in the village.

Tom said to himself that it was not such a hollow world, after all. He had discovered a great law of human action, without knowing it -- namely, that in order to make a man or a boy covet a thing, it is only necessary to make the

thing difficult to attain. If he had been a great and wise philosopher, like the writer of this book, he would now have comprehended that Work consists of whatever a body is obliged to do, and that Play consists of whatever a body is not obliged to do. And this would help him to understand why constructing artificial flowers or performing on a tread-mill is work, while rolling ten-pins or climbing Mont Blanc is only amusement. There are wealthy gentlemen in England who drive four-horse passenger-coaches twenty or thirty miles on a daily line, in the summer, because the privilege costs them considerable money; but if they were offered wages for the service, that would turn it into work and then they would resign.

The boy mused awhile over the substantial change which had taken place in his worldly circumstances, and then wended toward headquarters to report.

The Adventures of Tom Sawyer/Chapter III

Tom presented himself before Aunt Polly, who was sitting by an open window in a pleasant rearward apartment, which was bedroom, breakfast-room, dining-room, and library, combined. The balmy summer air, the restful quiet, the odor of the flowers, and the drowsing murmur of the bees had had their effect, and she was nodding over her knitting -- for she had no company but the cat, and it was asleep in her lap. Her spectacles were propped up on her gray head for safety. She had thought that of course Tom had deserted long ago, and she wondered at seeing him place himself in her power again in this intrepid way. He said: "Mayn't I go and play now, aunt?"

"What, a'ready? How much have you done?"

"It's all done, aunt."

"Tom, don't lie to me -- I can't bear it."

"I ain't, aunt; it is all done."

Aunt Polly placed small trust in such evidence. She went out to see for herself; and she would have been content to find twenty per cent. of Tom's statement true. When she found the entire fence whitewashed, and not only whitewashed but elaborately coated and recoated, and even a streak added to the ground, her astonishment was almost unspeakable. She said:

"Well, I never! There's no getting round it, you can work when you're a mind to, Tom." And then she diluted the compliment by adding, "But it's powerful seldom you're a mind to, I'm bound to say. Well, go 'long and play; but mind you get back some time in a week, or I'll tan you."

She was so overcome by the splendor of his achievement that she took him into the closet and selected a choice apple and delivered it to him, along with an improving lecture upon the added value and flavor a treat took to itself when it came without sin through virtuous effort. And while she closed with a happy Scriptural flourish, he "hooked" a doughnut.

Then he skipped out, and saw Sid just starting up the outside stairway that led to the back rooms on the second floor. Clods were handy and the air was full of them in a twinkling. They raged around Sid like a hail-storm; and before Aunt Polly could collect her surprised faculties and sally to the rescue, six or seven clods had taken personal effect, and Tom was over the fence and gone. There was a gate, but as a general thing he was too crowded for time to make use of it. His soul was at peace, now that he had settled with Sid for calling attention to his black thread and getting him into trouble.

Tom skirted the block, and came round into a muddy alley that led by the back of his aunt's cowstable. He presently got safely beyond the reach of capture and punishment, and hastened toward the public square of the village, where two "military" companies of boys had met for conflict, according to previous appointment. Tom was General of one of these armies, Joe Harper (a bosom friend) General of the other. These two great commanders did not condescend to fight in person -- that being better suited to the still smaller fry -- but sat together on an eminence and conducted

the field operations by orders delivered through aides-de-camp. Tom's army won a great victory, after a long and hard-fought battle. Then the dead were counted, prisoners exchanged, the terms of the next disagreement agreed upon, and the day for the necessary battle appointed; after which the armies fell into line and marched away, and Tom turned homeward alone.

As he was passing by the house where Jeff Thatcher lived, he saw a new girl in the garden -- a lovely little blue-eyed creature with yellow hair plaited into two long-tails, white summer frock and embroidered pantalettes. The fresh-crowned hero fell without firing a shot. A certain Amy Lawrence vanished out of his heart and left not even a memory of herself behind. He had thought he loved her to distraction; he had regarded his passion as adoration; and behold it was only a poor little evanescent partiality. He had been months winning her; she had confessed hardly a week ago; he had been the happiest and the proudest boy in the world only seven short days, and here in one instant of time she had gone out of his heart like a casual stranger whose visit is done.

He worshipped this new angel with furtive eye, till he saw that she had discovered him; then he pretended he did not know she was present, and began to "show off" in all sorts of absurd boyish ways, in order to win her admiration. He kept up this grotesque foolishness for some time; but by-and-by, while he was in the midst of some dangerous gymnastic performances, he glanced aside and saw that the little girl was wending her way toward the house. Tom came up to the fence and leaned on it, grieving, and hoping she would tarry yet awhile longer. She halted a moment on the steps and then moved toward the door. Tom heaved a great sigh as she put her foot on the threshold. But his face lit up, right away, for she tossed a pansy over the fence a moment before she disappeared.

The boy ran around and stopped within a foot or two of the flower, and then shaded his eyes with his hand and began to look down street as if he had discovered something of interest going on in that direction. Presently he picked up a straw and began trying to balance it on his nose, with his head tilted far back; and as he moved from side to side, in his efforts, he edged nearer and nearer toward the pansy; finally his bare foot rested upon it, his pliant toes closed upon it, and he hopped away with the treasure and disappeared round the corner. But only for a minute -- only while he could button the flower inside his jacket, next his heart -- or next his stomach, possibly, for he was not much posted in anatomy, and not hypercritical, anyway.

He returned, now, and hung about the fence till nightfall, "showing off," as before; but the girl never exhibited herself again, though Tom comforted himself a little with the hope that she had been near some window, meantime, and been aware of his attentions. Finally he strode home reluctantly, with his poor head full of visions.

All through supper his spirits were so high that his aunt wondered "what had got into the child." He took a good scolding about clodding Sid, and did not seem to mind it in the least. He tried to steal sugar under his aunt's very nose, and got his knuckles rapped for it. He said:

"Aunt, you don't whack Sid when he takes it."

"Well, Sid don't torment a body the way you do. You'd be always into that sugar if I warn't watching you."

Presently she stepped into the kitchen, and Sid, happy in his immunity, reached for the sugar-bowl -- a sort of glorying over Tom which was well-nigh unbearable. But Sid's fingers slipped and the bowl dropped and broke. Tom was in ecstasies. In such ecstasies that he even controlled his tongue and was silent. He said to himself that he would not speak a word, even when his aunt came in, but would sit perfectly still till she asked who did the mischief; and then he would tell, and there would be nothing so good in the world as to see that pet model "catch it." He was so brimful of exultation that he could hardly hold himself when the old lady came back and stood above the wreck discharging lightnings of wrath from over her spectacles. He said to himself, "Now it's coming!" And the next instant he was sprawling on the floor! The potent palm was uplifted to strike again when Tom cried out:

"Hold on, now, what 'er you belting me for? -- Sid broke it!"

Aunt Polly paused, perplexed, and Tom looked for healing pity. But when she got her tongue again, she only said:

"Umf! Well, you didn't get a lick amiss, I reckon. You been into some other audacious mischief when I wasn't around, like enough."

Then her conscience reproached her, and she yearned to say something kind and loving; but she judged that this would be construed into a confession that she had been in the wrong, and discipline forbade that. So she kept silence, and went about her affairs with a troubled heart. Tom sulked in a corner and exalted his woes. He knew that in her heart his aunt was on her knees to him, and he was morosely gratified by the consciousness of it. He would hang out no signals, he would take notice of none. He knew that a yearning glance fell upon him, now and then, through a film of tears, but he refused recognition of it. He pictured himself lying sick unto death and his aunt bending over him beseeching one little forgiving word, but he would turn his face to the wall, and die with that word unsaid. Ah, how would she feel then? And he pictured himself brought home from the river, dead, with his curls all wet, and his sore heart at rest. How she would throw herself upon him, and how her tears would fall like rain, and her lips pray God to give her back her boy and she would never, never abuse him any more! But he would lie there cold and white and make no sign -- a poor little sufferer, whose griefs were at an end. He so worked upon his feelings with the pathos of these dreams, that he had to keep swallowing, he was so like to choke; and his eyes swam in a blur of water, which overflowed when he winked, and ran down and trickled from the end of his nose. And such a luxury to him was this petting of his sorrows, that he could not bear to have any worldly cheeriness or any grating delight intrude upon it; it was too sacred for such contact; and so, presently, when his cousin Mary danced in, all alive with the joy of seeing home again after an age-long visit of one week to the country, he got up and moved in clouds and darkness out at one door as she brought song and sunshine in at the other.

He wandered far from the accustomed haunts of boys, and sought desolate places that were in harmony with his spirit. A log raft in the river invited him, and he seated himself on its outer edge and contemplated the dreary vastness of the stream, wishing, the while, that he could only be drowned, all at once and unconsciously, without undergoing the uncomfortable routine devised by nature. Then he thought of his flower. He got it out, crumpled and wilted, and it mightily increased his dismal felicity. He wondered if she would pity him if she knew? Would she cry, and wish that she had a right to put her arms around his neck and comfort him? Or would she turn coldly away like all the hollow world? This picture brought such an agony of pleasurable suffering that he worked it over and over again in his mind and set it up in new and varied lights, till he wore it threadbare. At last he rose up sighing and departed in the darkness.

About half-past nine or ten o'clock he came along the deserted street to where the Adored Unknown lived; he paused a moment; no sound fell upon his listening ear; a candle was casting a dull glow upon the curtain of a second-story window. Was the sacred presence there? He climbed the fence, threaded his stealthy way through the plants, till he stood under that window; he looked up at it long, and with emotion; then he laid him down on the ground under it, disposing himself upon his back, with his hands clasped upon his breast and holding his poor wilted flower. And thus he would die -- out in the cold world, with no shelter over his homeless head, no friendly hand to wipe the death-damps from his brow, no loving face to bend pityingly over him when the great agony came. And thus she would see him when she looked out upon the glad morning, and oh! would she drop one little tear upon his poor, lifeless form, would she heave one little sigh to see a bright young life so rudely blighted, so untimely cut down?

The window went up, a maid-servant's discordant voice profaned the holy calm, and a deluge of water drenched the prone martyr's remains!

The strangling hero sprang up with a relieving snort. There was a whiz as of a missile in the air, mingled with the murmur of a curse, a sound as of shivering glass followed, and a small, vague form went over the fence and shot away in the gloom.

Not long after, as Tom, all undressed for bed, was surveying his drenched garments by the light of a tallow dip, Sid woke up; but if he had any dim idea of making any "references to allusions," he thought better of it and held his peace, for there was danger in Tom's eye.

Tom turned in without the added vexation of prayers, and Sid made mental note of the omission.

The Adventures of Tom Sawyer/Chapter IV

The sun rose upon a tranquil world, and beamed down upon the peaceful village like a benediction. Breakfast over, Aunt Polly had family worship: it began with a prayer built from the ground up of solid courses of Scriptural quotations, welded together with a thin mortar of originality; and from the summit of this she delivered a grim chapter of the Mosaic Law, as from Sinai.

Then Tom girded up his loins, so to speak, and went to work to "get his verses." Sid had learned his lesson days before. Tom bent all his energies to the memorizing of five verses, and he chose part of the Sermon on the Mount, because he could find no verses that were shorter. At the end of half an hour Tom had a vague general idea of his lesson, but no more, for his mind was traversing the whole field of human thought, and his hands were busy with distracting recreations. Mary took his book to hear him recite, and he tried to find his way through the fog:

"Blessed are the -- a -- a --"

"Poor" --

"Yes -- poor; blessed are the poor -- a -- a --"

"In spirit --"

"In spirit; blessed are the poor in spirit, for they -- they --"

"Theirs --"

"For theirs. Blessed are the poor in spirit, for theirs is the kingdom of heaven. Blessed are they that mourn, for they -- they --"

"Sh --"

"For they -- a --"

"S, H, A --"

"For they S, H -- Oh, I don't know what it is!"

"Shall!"

"Oh, shall! for they shall -- for they shall -- a -- a -- shall mourn -- a -- a -- blessed are they that shall -- they that -- a -- they that shall mourn, for they shall -- a -- shall What? Why don't you tell me, Mary? -- what do you want to be so mean for?"

"Oh, Tom, you poor thick-headed thing, I'm not teasing you. I wouldn't do that. You must go and learn it again. Don't you be discouraged, Tom, you'll manage it -- and if you do, I'll give you something ever so nice. There, now, that's a good boy."

"All right! What is it, Mary, tell me what it is."

"Never you mind, Tom. You know if I say it's nice, it is nice."

"You bet you that's so, Mary. All right, I'll tackle it again."

And he did "tackle it again" -- and under the double pressure of curiosity and prospective gain he did it with such spirit that he accomplished a shining success. Mary gave him a brand-new "Barlow" knife worth twelve and a half cents; and the convulsion of delight that swept his system shook him to his foundations. True, the knife would not cut anything, but it was a "sure-enough" Barlow, and there was inconceivable grandeur in that -- though where the Western boys ever got the idea that such a weapon could possibly be counterfeited to its injury is an imposing mystery and will always remain so, perhaps. Tom contrived to scarify the cupboard with it, and was arranging to begin on the bureau, when he was called off to dress for Sunday-school.

Mary gave him a tin basin of water and a piece of soap, and he went outside the door and set the basin on a little bench there; then he dipped the soap in the water and laid it down; turned up his sleeves; poured out the water on the ground, gently, and then entered the kitchen and began to wipe his face diligently on the towel behind the door. But

Mary removed the towel and said:

"Now ain't you ashamed, Tom. You mustn't be so bad. Water won't hurt you."

Tom was a trifle disconcerted. The basin was refilled, and this time he stood over it a little while, gathering resolution; took in a big breath and began. When he entered the kitchen presently, with both eyes shut and groping for the towel with his hands, an honorable testimony of suds and water was dripping from his face. But when he emerged from the towel, he was not yet satisfactory, for the clean territory stopped short at his chin and his jaws, like a mask; below and beyond this line there was a dark expanse of unirrigated soil that spread downward in front and backward around his neck. Mary took him in hand, and when she was done with him he was a man and a brother, without distinction of color, and his saturated hair was neatly brushed, and its short curls wrought into a dainty and symmetrical general effect. [He privately smoothed out the curls, with labor and difficulty, and plastered his hair close down to his head; for he held curls to be effeminate, and his own filled his life with bitterness.] Then Mary got out a suit of his clothing that had been used only on Sundays during two years -- they were simply called his "other clothes" -- and so by that we know the size of his wardrobe. The girl "put him to rights" after he had dressed himself; she buttoned his neat roundabout up to his chin, turned his vast shirt collar down over his shoulders, brushed him off and crowned him with his speckled straw hat. He now looked exceedingly improved and uncomfortable. He was fully as uncomfortable as he looked; for there was a restraint about whole clothes and cleanliness that galled him. He hoped that Mary would forget his shoes, but the hope was blighted; she coated them thoroughly with tallow, as was the custom, and brought them out. He lost his temper and said he was always being made to do everything he didn't want to do. But Mary said, persuasively:

"Please, Tom -- that's a good boy."

So he got into the shoes snarling. Mary was soon ready, and the three children set out for Sunday-school -- a place that Tom hated with his whole heart; but Sid and Mary were fond of it.

Sabbath-school hours were from nine to half-past ten; and then church service. Two of the children always remained for the sermon voluntarily, and the other always remained too -- for stronger reasons. The church's high-backed, uncushioned pews would seat about three hundred persons; the edifice was but a small, plain affair, with a sort of pine board tree-box on top of it for a steeple. At the door Tom dropped back a step and accosted a Sunday-dressed comrade:

"Say, Billy, got a yaller ticket?"

"Yes."

"What'll you take for her?"

"What'll you give?"

"Piece of lickrish and a fish-hook."

"Less see 'em."

Tom exhibited. They were satisfactory, and the property changed hands. Then Tom traded a couple of white alleys for three red tickets, and some small trifle or other for a couple of blue ones. He waylaid other boys as they came, and went on buying tickets of various colors ten or fifteen minutes longer. He entered the church, now, with a swarm of clean and noisy boys and girls, proceeded to his seat and started a quarrel with the first boy that came handy. The teacher, a grave, elderly man, interfered; then turned his back a moment and Tom pulled a boy's hair in the next bench, and was absorbed in his book when the boy turned around; stuck a pin in another boy, presently, in order to hear him say "Ouch!" and got a new reprimand from his teacher. Tom's whole class were of a pattern -- restless, noisy, and troublesome. When they came to recite their lessons, not one of them knew his verses perfectly, but had to be prompted all along. However, they worried through, and each got his reward -- in small blue tickets, each with a passage of Scripture on it; each blue ticket was pay for two verses of the recitation. Ten blue tickets equalled a red one, and could be exchanged for it; ten red tickets equalled a yellow one; for ten yellow tickets the superintendent gave a very plainly bound Bible (worth forty cents in those easy times) to the pupil. How many of my readers would

have the industry and application to memorize two thousand verses, even for a Dore Bible? And yet Mary had acquired two Bibles in this way -- it was the patient work of two years -- and a boy of German parentage had won four or five. He once recited three thousand verses without stopping; but the strain upon his mental faculties was too great, and he was little better than an idiot from that day forth -- a grievous misfortune for the school, for on great occasions, before company, the superintendent (as Tom expressed it) had always made this boy come out and "spread himself." Only the older pupils managed to keep their tickets and stick to their tedious work long enough to get a Bible, and so the delivery of one of these prizes was a rare and noteworthy circumstance; the successful pupil was so great and conspicuous for that day that on the spot every scholar's heart was fired with a fresh ambition that often lasted a couple of weeks. It is possible that Tom's mental stomach had never really hungered for one of those prizes, but unquestionably his entire being had for many a day longed for the glory and the eclat that came with it.

In due course the superintendent stood up in front of the pulpit, with a closed hymn-book in his hand and his forefinger inserted between its leaves, and commanded attention. When a Sunday-school superintendent makes his customary little speech, a hymn-book in the hand is as necessary as is the inevitable sheet of music in the hand of a singer who stands forward on the platform and sings a solo at a concert -- though why, is a mystery: for neither the hymn-book nor the sheet of music is ever referred to by the sufferer. This superintendent was a slim creature of thirty-five, with a sandy goatee and short sandy hair; he wore a stiff standing-collar whose upper edge almost reached his ears and whose sharp points curved forward abreast the corners of his mouth -- a fence that compelled a straight lookout ahead, and a turning of the whole body when a side view was required; his chin was propped on a spreading cravat which was as broad and as long as a bank-note, and had fringed ends; his boot toes were turned sharply up, in the fashion of the day, like sleigh-runners -- an effect patiently and laboriously produced by the young men by sitting with their toes pressed against a wall for hours together. Mr. Walters was very earnest of mien, and very sincere and honest at heart; and he held sacred things and places in such reverence, and so separated them from worldly matters, that unconsciously to himself his Sunday-school voice had acquired a peculiar intonation which was wholly absent on week-days. He began after this fashion:

"Now, children, I want you all to sit up just as straight and pretty as you can and give me all your attention for a minute or two. There -- that is it. That is the way good little boys and girls should do. I see one little girl who is looking out of the window -- I am afraid she thinks I am out there somewhere -- perhaps up in one of the trees making a speech to the little birds. [Applausive titter.] I want to tell you how good it makes me feel to see so many bright, clean little faces assembled in a place like this, learning to do right and be good." And so forth and so on. It is not necessary to set down the rest of the oration. It was of a pattern which does not vary, and so it is familiar to us all.

The latter third of the speech was marred by the resumption of fights and other recreations among certain of the bad boys, and by fidgetings and whisperings that extended far and wide, washing even to the bases of isolated and incorruptible rocks like Sid and Mary. But now every sound ceased suddenly, with the subsidence of Mr. Walters' voice, and the conclusion of the speech was received with a burst of silent gratitude.

A good part of the whispering had been occasioned by an event which was more or less rare -- the entrance of visitors: lawyer Thatcher, accompanied by a very feeble and aged man; a fine, portly, middle-aged gentleman with iron-gray hair; and a dignified lady who was doubtless the latter's wife. The lady was leading a child. Tom had been restless and full of chafings and repinings; conscience-smitten, too -- he could not meet Amy Lawrence's eye, he could not brook her loving gaze. But when he saw this small new-comer his soul was all ablaze with bliss in a moment. The next moment he was "showing off" with all his might -- cuffing boys, pulling hair, making faces -- in a word, using every art that seemed likely to fascinate a girl and win her applause. His exaltation had but one alloy -- the memory of his humiliation in this angel's garden -- and that record in sand was fast washing out, under the waves of happiness that were sweeping over it now.

The visitors were given the highest seat of honor, and as soon as Mr. Walters' speech was finished, he introduced them to the school. The middle-aged man turned out to be a prodigious personage -- no less a one than the county

judge -- altogether the most august creation these children had ever looked upon -- and they wondered what kind of material he was made of -- and they half wanted to hear him roar, and were half afraid he might, too. He was from Constantinople, twelve miles away -- so he had travelled, and seen the world -- these very eyes had looked upon the county court-house -- which was said to have a tin roof. The awe which these reflections inspired was attested by the impressive silence and the ranks of staring eyes. This was the great Judge Thatcher, brother of their own lawyer. Jeff Thatcher immediately went forward, to be familiar with the great man and be envied by the school. It would have been music to his soul to hear the whisperings:

"Look at him, Jim! He's a going up there. Say -- look! he's a going to shake hands with him -- he is shaking hands with him! By jings, don't you wish you was Jeff?"

Mr. Walters fell to "showing off," with all sorts of official bustlings and activities, giving orders, delivering judgments, discharging directions here, there, everywhere that he could find a target. The librarian "showed off" -- running hither and thither with his arms full of books and making a deal of the splutter and fuss that insect authority delights in. The young lady teachers "showed off" -- bending sweetly over pupils that were lately being boxed, lifting pretty warning fingers at bad little boys and patting good ones lovingly. The young gentlemen teachers "showed off" with small scoldings and other little displays of authority and fine attention to discipline -- and most of the teachers, of both sexes, found business up at the library, by the pulpit; and it was business that frequently had to be done over again two or three times (with much seeming vexation). The little girls "showed off" in various ways, and the little boys "showed off" with such diligence that the air was thick with paper wads and the murmur of scufflings. And above it all the great man sat and beamed a majestic judicial smile upon all the house, and warmed himself in the sun of his own grandeur -- for he was "showing off," too.

There was only one thing wanting to make Mr. Walters' ecstasy complete, and that was a chance to deliver a Bible-prize and exhibit a prodigy. Several pupils had a few yellow tickets, but none had enough -- he had been around among the star pupils inquiring. He would have given worlds, now, to have that German lad back again with a sound mind.

And now at this moment, when hope was dead, Tom Sawyer came forward with nine yellow tickets, nine red tickets, and ten blue ones, and demanded a Bible. This was a thunderbolt out of a clear sky. Walters was not expecting an application from this source for the next ten years. But there was no getting around it -- here were the certified checks, and they were good for their face. Tom was therefore elevated to a place with the Judge and the other elect, and the great news was announced from headquarters. It was the most stunning surprise of the decade, and so profound was the sensation that it lifted the new hero up to the judicial one's altitude, and the school had two marvels to gaze upon in place of one. The boys were all eaten up with envy -- but those that suffered the bitterest pangs were those who perceived too late that they themselves had contributed to this hated splendor by trading tickets to Tom for the wealth he had amassed in selling whitewashing privileges. These despised themselves, as being the dupes of a wily fraud, a guileful snake in the grass.

The prize was delivered to Tom with as much effusion as the superintendent could pump up under the circumstances; but it lacked somewhat of the true gush, for the poor fellow's instinct taught him that there was a mystery here that could not well bear the light, perhaps; it was simply preposterous that this boy had warehoused two thousand sheaves of Scriptural wisdom on his premises -- a dozen would strain his capacity, without a doubt.

Amy Lawrence was proud and glad, and she tried to make Tom see it in her face -- but he wouldn't look. She wondered; then she was just a grain troubled; next a dim suspicion came and went -- came again; she watched; a furtive glance told her worlds -- and then her heart broke, and she was jealous, and angry, and the tears came and she hated everybody. Tom most of all (she thought).

Tom was introduced to the Judge; but his tongue was tied, his breath would hardly come, his heart quaked -- partly because of the awful greatness of the man, but mainly because he was her parent. He would have liked to fall down and worship him, if it were in the dark. The Judge put his hand on Tom's head and called him a fine little man, and asked him what his name was. The boy stammered, gasped, and got it out:

"Tom."

"Oh, no, not Tom -- it is --"

"Thomas."

"Ah, that's it. I thought there was more to it, maybe. That's very well. But you've another one I daresay, and you'll tell it to me, won't you?"

"Tell the gentleman your other name, Thomas," said Walters, "and say sir. You mustn't forget your manners."

"Thomas Sawyer -- sir."

"That's it! That's a good boy. Fine boy. Fine, manly little fellow. Two thousand verses is a great many -- very, very great many. And you never can be sorry for the trouble you took to learn them; for knowledge is worth more than anything there is in the world; it's what makes great men and good men; you'll be a great man and a good man yourself, some day, Thomas, and then you'll look back and say, It's all owing to the precious Sunday-school privileges of my boyhood -- it's all owing to my dear teachers that taught me to learn -- it's all owing to the good superintendent, who encouraged me, and watched over me, and gave me a beautiful Bible -- a splendid elegant Bible -- to keep and have it all for my own, always -- it's all owing to right bringing up! That is what you will say, Thomas -- and you wouldn't take any money for those two thousand verses -- no indeed you wouldn't. And now you wouldn't mind telling me and this lady some of the things you've learned -- no, I know you wouldn't -- for we are proud of little boys that learn. Now, no doubt you know the names of all the twelve disciples. Won't you tell us the names of the first two that were appointed?"

Tom was tugging at a button-hole and looking sheepish. He blushed, now, and his eyes fell. Mr. Walters' heart sank within him. He said to himself, it is not possible that the boy can answer the simplest question -- why did the Judge ask him? Yet he felt obliged to speak up and say:

"Answer the gentleman, Thomas -- don't be afraid."

Tom still hung fire.

"Now I know you'll tell me," said the lady. "The names of the first two disciples were --"

"David and Goliath!"

Let us draw the curtain of charity over the rest of the scene.

The Adventures of Tom Sawyer/Chapter V

About half-past ten the cracked bell of the small church began to ring, and presently the people began to gather for the morning sermon. The Sunday-school children distributed themselves about the house and occupied pews with their parents, so as to be under supervision. Aunt Polly came, and Tom and Sid and Mary sat with her -- Tom being placed next the aisle, in order that he might be as far away from the open window and the seductive outside summer scenes as possible. The crowd filed up the aisles: the aged and needy postmaster, who had seen better days; the mayor and his wife -- for they had a mayor there, among other unnecessaries; the justice of the peace; the widow Douglass, fair, smart, and forty, a generous, good-hearted soul and well-to-do, her hill mansion the only palace in the town, and the most hospitable and much the most lavish in the matter of festivities that St. Petersburg could boast; the bent and venerable Major and Mrs. Ward; lawyer Riverson, the new notable from a distance; next the belle of the village, followed by a troop of lawn-clad and ribbon-decked young heart-breakers; then all the young clerks in town in a body -- for they had stood in the vestibule sucking their cane-heads, a circling wall of oiled and simpering admirers, till the last girl had run their gantlet; and last of all came the Model Boy, Willie Mufferson, taking as heedful care of his mother as if she were cut glass. He always brought his mother to church, and was the pride of all the matrons. The boys all hated him, he was so good. And besides, he had been "thrown up to them" so much. His white handkerchief was hanging out of his pocket behind, as usual on Sundays -- accidentally. Tom had no handkerchief, and he looked upon boys who had as snobs.

The congregation being fully assembled, now, the bell rang once more, to warn laggards and stragglers, and then a solemn hush fell upon the church which was only broken by the tittering and whispering of the choir in the gallery. The choir always tittered and whispered all through service. There was once a church choir that was not ill-bred, but I have forgotten where it was, now. It was a great many years ago, and I can scarcely remember anything about it, but I think it was in some foreign country.

The minister gave out the hymn, and read it through with a relish, in a peculiar style which was much admired in that part of the country. His voice began on a medium key and climbed steadily up till it reached a certain point, where it bore with strong emphasis upon the topmost word and then plunged down as if from a spring-board:

Shall I be car-ri-ed toe the skies, on flow'ry beds
of ease,
Whilst others fight to win the prize, and sail thro' bloody seas?

He was regarded as a wonderful reader. At church "sociables" he was always called upon to read poetry; and when he was through, the ladies would lift up their hands and let them fall helplessly in their laps, and "wall" their eyes, and shake their heads, as much as to say, "Words cannot express it; it is too beautiful, too beautiful for this mortal earth." After the hymn had been sung, the Rev. Mr. Sprague turned himself into a bulletin-board, and read off "notices" of meetings and societies and things till it seemed that the list would stretch out to the crack of doom -- a queer custom which is still kept up in America, even in cities, away here in this age of abundant newspapers. Often, the less there is to justify a traditional custom, the harder it is to get rid of it.

And now the minister prayed. A good, generous prayer it was, and went into details: it pleaded for the church, and the little children of the church; for the other churches of the village; for the village itself; for the county; for the State; for the State officers; for the United States; for the churches of the United States; for Congress; for the President; for the officers of the Government; for poor sailors, tossed by stormy seas; for the oppressed millions groaning under the heel of European monarchies and Oriental despotisms; for such as have the light and the good tidings, and yet have not eyes to see nor ears to hear withal; for the heathen in the far islands of the sea; and closed with a supplication that the words he was about to speak might find grace and favor, and be as seed sown in fertile ground, yielding in time a grateful harvest of good. Amen.

There was a rustling of dresses, and the standing congregation sat down. The boy whose history this book relates did not enjoy the prayer, he only endured it -- if he even did that much. He was restive all through it; he kept tally of the details of the prayer, unconsciously -- for he was not listening, but he knew the ground of old, and the clergyman's regular route over it -- and when a little trifle of new matter was interlarded, his ear detected it and his whole nature resented it; he considered additions unfair, and scoundrelly. In the midst of the prayer a fly had lit on the back of the pew in front of him and tortured his spirit by calmly rubbing its hands together, embracing its head with its arms, and polishing it so vigorously that it seemed to almost part company with the body, and the slender thread of a neck was exposed to view; scraping its wings with its hind legs and smoothing them to its body as if they had been coat-tails; going through its whole toilet as tranquilly as if it knew it was perfectly safe. As indeed it was; for as sorely as Tom's hands itched to grab for it they did not dare -- he believed his soul would be instantly destroyed if he did such a thing while the prayer was going on. But with the closing sentence his hand began to curve and steal forward; and the instant the "Amen" was out the fly was a prisoner of war. His aunt detected the act and made him let it go.

The minister gave out his text and droned along monotonously through an argument that was so prosy that many a head by and by began to nod -- and yet it was an argument that dealt in limitless fire and brimstone and thinned the predestined elect down to a company so small as to be hardly worth the saving. Tom counted the pages of the sermon; after church he always knew how many pages there had been, but he seldom knew anything else about the discourse. However, this time he was really interested for a little while. The minister made a grand and moving picture of the assembling together of the world's hosts at the millennium when the lion and the lamb should lie down together and a little child should lead them. But the pathos, the lesson, the moral of the great spectacle were lost upon the boy; he only thought of the conspicuousness of the principal character before the on-looking nations; his face lit with the thought, and he said to himself that he wished he could be that child, if it was a tame lion.

Now he lapsed into suffering again, as the dry argument was resumed. Presently he bethought him of a treasure he had and got it out. It was a large black beetle with formidable jaws -- a "pinchbug," he called it. It was in a percussion-cap box. The first thing the beetle did was to take him by the finger. A natural fillip followed, the beetle went floundering into the aisle and lit on its back, and the hurt finger went into the boy's mouth. The beetle lay there working its helpless legs, unable to turn over. Tom eyed it, and longed for it; but it was safe out of his reach. Other people uninterested in the sermon found relief in the beetle, and they eyed it too. Presently a vagrant poodle dog came idling along, sad at heart, lazy with the summer softness and the quiet, weary of captivity, sighing for change. He spied the beetle; the drooping tail lifted and wagged. He surveyed the prize; walked around it; smelt at it from a safe distance; walked around it again; grew bolder, and took a closer smell; then lifted his lip and made a gingerly snatch at it, just missing it; made another, and another; began to enjoy the diversion; subsided to his stomach with the beetle between his paws, and continued his experiments; grew weary at last, and then indifferent and absent-minded. His head nodded, and little by little his chin descended and touched the enemy, who seized it. There was a sharp yelp, a flirt of the poodle's head, and the beetle fell a couple of yards away, and lit on its back once more. The neighboring spectators shook with a gentle inward joy, several faces went behind fans and handkerchiefs, and Tom was entirely happy. The dog looked foolish, and probably felt so; but there was resentment in his heart, too, and a craving for revenge. So he went to the beetle and began a wary attack on it again; jumping at it from every point of a circle, lighting with his fore-paws within an inch of the creature, making even closer snatches at it with his teeth, and jerking his head till his ears flapped again. But he grew tired once more, after a while; tried to amuse himself with a fly but found no relief; followed an ant around, with his nose close to the floor, and quickly wearied of that; yawned, sighed, forgot the beetle entirely, and sat down on it. Then there was a wild yelp of agony and the poodle went sailing up the aisle; the yelps continued, and so did the dog; he crossed the house in front of the altar; he flew down the other aisle; he crossed before the doors; he clamored up the home-stretch; his anguish grew with his progress, till presently he was but a woolly comet moving in its orbit with the gleam and the speed of light. At last the frantic sufferer sheered from its course, and sprang into its master's lap; he flung it out of the window, and the voice of distress quickly thinned away and died in the distance.

By this time the whole church was red-faced and suffocating with suppressed laughter, and the sermon had come to a dead standstill. The discourse was resumed presently, but it went lame and halting, all possibility of impressiveness being at an end; for even the gravest sentiments were constantly being received with a smothered burst of unholy mirth, under cover of some remote pew-back, as if the poor parson had said a rarely facetious thing. It was a genuine relief to the whole congregation when the ordeal was over and the benediction pronounced.

Tom Sawyer went home quite cheerful, thinking to himself that there was some satisfaction about divine service when there was a bit of variety in it. He had but one marring thought; he was willing that the dog should play with his pinchbug, but he did not think it was upright in him to carry it off.

The Adventures of Tom Sawyer/Chapter VI

Monday morning found Tom Sawyer miserable. Monday morning always found him so -- because it began another week's slow suffering in school. He generally began that day with wishing he had had no intervening holiday, it made the going into captivity and fetters again so much more odious.

Tom lay thinking. Presently it occurred to him that he wished he was sick; then he could stay home from school. Here was a vague possibility. He canvassed his system. No ailment was found, and he investigated again. This time he thought he could detect colicky symptoms, and he began to encourage them with considerable hope. But they soon grew feeble, and presently died wholly away. He reflected further. Suddenly he discovered something. One of his upper front teeth was loose. This was lucky; he was about to begin to groan, as a "starter," as he called it, when it occurred to him that if he came into court with that argument, his aunt would pull it out, and that would hurt. So he thought he would hold the tooth in reserve for the present, and seek further. Nothing offered for some little time, and then he remembered hearing the doctor tell about a certain thing that laid up a patient for two or three weeks and threatened to make him lose a finger. So the boy eagerly drew his sore toe from under the sheet and held it up for inspection. But now he did not know the necessary symptoms. However, it seemed well worth while to chance it, so he fell to groaning with considerable spirit.

But Sid slept on unconscious.

Tom groaned louder, and fancied that he began to feel pain in the toe.

No result from Sid.

Tom was panting with his exertions by this time. He took a rest and then swelled himself up and fetched a succession of admirable groans.

Sid snored on.

Tom was aggravated. He said, "Sid, Sid!" and shook him. This course worked well, and Tom began to groan again. Sid yawned, stretched, then brought himself up on his elbow with a snort, and began to stare at Tom. Tom went on groaning. Sid said:

"Tom! Say, Tom!" [No response.] "Here, Tom! Tom! What is the matter, Tom?" And he shook him and looked in his face anxiously.

Tom moaned out:

"Oh, don't, Sid. Don't joggle me."

"Why, what's the matter, Tom? I must call auntie."

"No -- never mind. It'll be over by and by, maybe. Don't call anybody."

"But I must! Don't groan so, Tom, it's awful. How long you been this way?"

"Hours. Ouch! Oh, don't stir so, Sid, you'll kill me."

"Tom, why didn't you wake me sooner ? Oh, Tom, Don't! It makes my flesh crawl to hear you. Tom, what is the matter?"

"I forgive you everything, Sid. [Groan.] Everything you've ever done to me. When I'm gone --"

"Oh, Tom, you ain't dying, are you? Don't, Tom -- oh, don't. Maybe --"

"I forgive everybody, Sid. [Groan.] Tell 'em so, Sid. And Sid, you give my window-sash and my cat with one eye to that new girl that's come to town, and tell her --"

But Sid had snatched his clothes and gone. Tom was suffering in reality, now, so handsomely was his imagination working, and so his groans had gathered quite a genuine tone.

Sid flew down-stairs and said:

"Oh, Aunt Polly, come! Tom's dying!"

"Dying!"

"Yes'm. Don't wait -- come quick!"

"Rubbage! I don't believe it!"

But she fled up-stairs, nevertheless, with Sid and Mary at her heels. And her face grew white, too, and her lip trembled. When she reached the bedside she gasped out:

"You, Tom! Tom, what's the matter with you?"

"Oh, auntie, I'm --"

"What's the matter with you -- what is the matter with you, child?"

"Oh, auntie, my sore toe's mortified!"

The old lady sank down into a chair and laughed a little, then cried a little, then did both together. This restored her and she said:

"Tom, what a turn you did give me. Now you shut up that nonsense and climb out of this."

The groans ceased and the pain vanished from the toe. The boy felt a little foolish, and he said:

"Aunt Polly, it seemed mortified, and it hurt so I never minded my tooth at all."

"Your tooth, indeed! What's the matter with your tooth?"

"One of them's loose, and it aches perfectly awful."

"There, there, now, don't begin that groaning again. Open your mouth. Well -- your tooth is loose, but you're not going to die about that. Mary, get me a silk thread, and a chunk of fire out of the kitchen."

Tom said:

"Oh, please, auntie, don't pull it out. It don't hurt any more. I wish I may never stir if it does. Please don't, auntie. I don't want to stay home from school."

"Oh, you don't, don't you? So all this row was because you thought you'd get to stay home from school and go a-fishing? Tom, Tom, I love you so, and you seem to try every way you can to break my old heart with your outrageousness." By this time the dental instruments were ready. The old lady made one end of the silk thread fast to Tom's tooth with a loop and tied the other to the bedpost. Then she seized the chunk of fire and suddenly thrust it almost into the boy's face. The tooth hung dangling by the bedpost, now.

But all trials bring their compensations. As Tom wended to school after breakfast, he was the envy of every boy he met because the gap in his upper row of teeth enabled him to expectorate in a new and admirable way. He gathered quite a following of lads interested in the exhibition; and one that had cut his finger and had been a centre of fascination and homage up to this time, now found himself suddenly without an adherent, and shorn of his glory. His heart was heavy, and he said with a disdain which he did not feel that it wasn't anything to spit like Tom Sawyer; but another boy said, "Sour grapes!" and he wandered away a dismantled hero.

Shortly Tom came upon the juvenile pariah of the village, Huckleberry Finn, son of the town drunkard. Huckleberry was cordially hated and dreaded by all the mothers of the town, because he was idle and lawless and vulgar and bad -- and because all their children admired him so, and delighted in his forbidden society, and wished they dared to be like him. Tom was like the rest of the respectable boys, in that he envied Huckleberry his gaudy outcast condition, and was under strict orders not to play with him. So he played with him every time he got a chance. Huckleberry was always dressed in the cast-off clothes of full-grown men, and they were in perennial bloom and fluttering with rags. His hat was a vast ruin with a wide crescent lopped out of its brim; his coat, when he wore one, hung nearly to his heels and had the rearward buttons far down the back; but one suspender supported his trousers; the seat of the trousers bagged low and contained nothing, the fringed legs dragged in the dirt when not rolled up.

Huckleberry came and went, at his own free will. He slept on doorsteps in fine weather and in empty hogsheads in wet; he did not have to go to school or to church, or call any being master or obey anybody; he could go fishing or swimming when and where he chose, and stay as long as it suited him; nobody forbade him to fight; he could sit up as late as he pleased; he was always the first boy that went barefoot in the spring and the last to resume leather in the fall; he never had to wash, nor put on clean clothes; he could swear wonderfully. In a word, everything that goes to make life precious that boy had. So thought every harassed, hampered, respectable boy in St. Petersburg.

Tom hailed the romantic outcast:

"Hello, Huckleberry!"

"Hello yourself, and see how you like it."

"What's that you got?"

"Dead cat."

"Lemme see him, Huck. My, he's pretty stiff. Where'd you get him?"

"Bought him off'n a boy."

"What did you give?"

"I give a blue ticket and a bladder that I got at the slaughter-house."

"Where'd you get the blue ticket?"

"Bought it off'n Ben Rogers two weeks ago for a hoop-stick."

"Say -- what is dead cats good for, Huck?"

"Good for? Cure warts with."

"No! Is that so? I know something that's better."

"I bet you don't. What is it?"

"Why, spunk-water."

"Spunk-water! I wouldn't give a dern for spunk-water."

"You wouldn't, wouldn't you? D'you ever try it?"

"No, I hain't. But Bob Tanner did."

"Who told you so!"

"Why, he told Jeff Thatcher, and Jeff told Johnny Baker, and Johnny told Jim Hollis, and Jim told Ben Rogers, and Ben told a nigger, and the nigger told me. There now!"

"Well, what of it? They'll all lie. Leastways all but the nigger. I don't know him. But I never see a nigger that wouldn't lie. Shucks! Now you tell me how Bob Tanner done it, Huck."

"Why, he took and dipped his hand in a rotten stump where the rain-water was."

"In the daytime?"

"Certainly."

"With his face to the stump?"

"Yes. Least I reckon so."

"Did he say anything?"

"I don't reckon he did. I don't know."

"Aha! Talk about trying to cure warts with spunk-water such a blame fool way as that! Why, that ain't a-going to do any good. You got to go all by yourself, to the middle of the woods, where you know there's a spunk-water stump, and just as it's midnight you back up against the stump and jam your hand in and say:

'Barley-corn, barley-corn, injun-meal shorts,
Spunk-water, spunk-water, swaller these warts, '

and then walk away quick, eleven steps, with your eyes shut, and then turn around three times and walk home without speaking to anybody. Because if you speak the charm's busted."

"Well, that sounds like a good way; but that ain't the way Bob Tanner done."

"No, sir, you can bet he didn't, becuz he's the wartiest boy in this town; and he wouldn't have a wart on him if he'd knowed how to work spunk-water. I've took off thousands of warts off of my hands that way, Huck. I play with frogs so much that I've always got considerable many warts. Sometimes I take 'em off with a bean."

"Yes, bean's good. I've done that."

"Have you? What's your way?"

"You take and split the bean, and cut the wart so as to get some blood, and then you put the blood on one piece of the bean and take and dig a hole and bury it 'bout midnight at the crossroads in the dark of the moon, and then you burn up the rest of the bean. You see that piece that's got the blood on it will keep drawing and drawing, trying to fetch the other piece to it, and so that helps the blood to draw the wart, and pretty soon off she comes."

"Yes, that's it, Huck -- that's it; though when you're burying it if you say 'Down bean; off wart; come no more to bother me!' it's better. That's the way Joe Harper does, and he's been nearly to Coonville and most everywhere. But say -- how do you cure 'em with dead cats?"

"Why, you take your cat and go and get in the graveyard 'long about midnight when somebody that was wicked has been buried; and when it's midnight a devil will come, or maybe two or three, but you can't see 'em, you can only hear something like the wind, or maybe hear 'em talk; and when they're taking that feller away, you heave your cat after 'em and say, 'Devil follow corpse, cat follow devil, warts follow cat, I'm done with ye!' That'll fetch any wart."

"Sounds right. D'you ever try it, Huck?"

"No, but old Mother Hopkins told me."

"Well, I reckon it's so, then. Becuz they say she's a witch."

"Say! Why, Tom, I know she is. She witched pap. Pap says so his own self. He come along one day, and he see she was a-witching him, so he took up a rock, and if she hadn't dodged, he'd a got her. Well, that very night he rolled off'n a shed wher' he was a layin drunk, and broke his arm."

"Why, that's awful. How did he know she was a-witching him?"

"Lord, pap can tell, easy. Pap says when they keep looking at you right stiddy, they're a-witching you. Specially if they mumble. Becuz when they mumble they're saying the Lord's Prayer backwards."

"Say, Hucky, when you going to try the cat?"

"To-night. I reckon they'll come after old Hoss Williams to-night."

"But they buried him Saturday. Didn't they get him Saturday night?"

"Why, how you talk! How could their charms work till midnight? -- and then it's Sunday. Devils don't slosh around much of a Sunday, I don't reckon."

"I never thought of that. That's so. Lemme go with you?"

"Of course -- if you ain't afeard."

"Afeard! 'Tain't likely. Will you meow?"

"Yes -- and you meow back, if you get a chance. Last time, you kep' me a-meowing around till old Hays went to throwing rocks at me and says 'Dern that cat!' and so I hove a brick through his window -- but don't you tell."

"I won't. I couldn't meow that night, becuz auntie was watching me, but I'll meow this time. Say -- what's that?"

"Nothing but a tick."

"Where'd you get him?"

"Out in the woods."

"What'll you take for him?"

"I don't know. I don't want to sell him."

"All right. It's a mighty small tick, anyway."

"Oh, anybody can run a tick down that don't belong to them. I'm satisfied with it. It's a good enough tick for me."

"Sho, there's ticks a plenty. I could have a thousand of 'em if I wanted to."

"Well, why don't you? Becuz you know mighty well you can't. This is a pretty early tick, I reckon. It's the first one I've seen this year."

"Say, Huck -- I'll give you my tooth for him."

"Less see it."

Tom got out a bit of paper and carefully unrolled it. Huckleberry viewed it wistfully. The temptation was very strong. At last he said:

"Is it genuwyne?"

Tom lifted his lip and showed the vacancy.

"Well, all right," said Huckleberry, "it's a trade."

Tom enclosed the tick in the percussion-cap box that had lately been the pinchbug's prison, and the boys separated, each feeling wealthier than before.

When Tom reached the little isolated frame schoolhouse, he strode in briskly, with the manner of one who had come with all honest speed. He hung his hat on a peg and flung himself into his seat with business-like alacrity. The master, throned on high in his great splint-bottom arm-chair, was dozing, lulled by the drowsy hum of study. The interruption roused him.

"Thomas Sawyer!"

Tom knew that when his name was pronounced in full, it meant trouble.

"Sir!"

"Come up here. Now, sir, why are you late again, as usual?"

Tom was about to take refuge in a lie, when he saw two long tails of yellow hair hanging down a back that he recognized by the electric sympathy of love; and by that form was the only vacant place on the girls' side of the school-house. He instantly said:

"I stopped to talk with Huckleberry Finn!"

The master's pulse stood still, and he stared helplessly. The buzz of study ceased. The pupils wondered if this foolhardy boy had lost his mind. The master said:

"You -- you did what?"

"Stopped to talk with Huckleberry Finn."

There was no mistaking the words.

"Thomas Sawyer, this is the most astounding confession I have ever listened to. No mere ferule will answer for this offence. Take off your jacket."

The master's arm performed until it was tired and the stock of switches notably diminished. Then the order followed:

"Now, sir, go and sit with the girls! And let this be a warning to you."

The titter that rippled around the room appeared to abash the boy, but in reality that result was caused rather more by his worshipful awe of his unknown idol and the dread pleasure that lay in his high good fortune. He sat down upon the end of the pine bench and the girl hitched herself away from him with a toss of her head. Nudges and winks and whispers traversed the room, but Tom sat still, with his arms upon the long, low desk before him, and seemed to study his book.

By and by attention ceased from him, and the accustomed school murmur rose upon the dull air once more. Presently the boy began to steal furtive glances at the girl. She observed it, "made a mouth" at him and gave him the back of her head for the space of a minute. When she cautiously faced around again, a peach lay before her. She thrust it away. Tom gently put it back. She thrust it away again, but with less animosity. Tom patiently returned it to its place. Then she let it remain. Tom scrawled on his slate, "Please take it -- I got more." The girl glanced at the words, but made no sign. Now the boy began to draw something on the slate, hiding his work with his left hand. For a time the girl refused to notice; but her human curiosity presently began to manifest itself by hardly perceptible signs. The boy worked on, apparently unconscious. The girl made a sort of noncommittal attempt to see, but the boy did not betray that he was aware of it. At last she gave in and hesitatingly whispered:

"Let me see it."

Tom partly uncovered a dismal caricature of a house with two gable ends to it and a corkscrew of smoke issuing from the chimney. Then the girl's interest began to fasten itself upon the work and she forgot everything else. When it was finished, she gazed a moment, then whispered:

"It's nice -- make a man."

The artist erected a man in the front yard, that resembled a derrick. He could have stepped over the house; but the girl was not hypercritical; she was satisfied with the monster, and whispered:

"It's a beautiful man -- now make me coming along."

Tom drew an hour-glass with a full moon and straw limbs to it and armed the spreading fingers with a portentous fan. The girl said:

"It's ever so nice -- I wish I could draw."

"It's easy," whispered Tom, "I'll learn you."

"Oh, will you? When?"

"At noon. Do you go home to dinner?"

"I'll stay if you will."

"Good -- that's a whack. What's your name?"

"Becky Thatcher. What's yours? Oh, I know. It's Thomas Sawyer."

"That's the name they lick me by. I'm Tom when I'm good. You call me Tom, will you?"

"Yes."

Now Tom began to scrawl something on the slate, hiding the words from the girl. But she was not backward this time. She begged to see. Tom said:

"Oh, it ain't anything."

"Yes it is."

"No it ain't. You don't want to see."

"Yes I do, indeed I do. Please let me."

"You'll tell."

"No I won't -- deed and deed and double deed won't."

"You won't tell anybody at all? Ever, as long as you live?"

"No, I won't ever tell anybody. Now let me."

"Oh, you don't want to see!"

"Now that you treat me so, I will see." And she put her small hand upon his and a little scuffle ensued, Tom pretending to resist in earnest but letting his hand slip by degrees till these words were revealed: "I love You."

"Oh, you bad thing!" And she hit his hand a smart rap, but reddened and looked pleased, nevertheless.

Just at this juncture the boy felt a slow, fateful grip closing on his ear, and a steady lifting impulse. In that vise he was borne across the house and deposited in his own seat, under a peppering fire of giggles from the whole school. Then the master stood over him during a few awful moments, and finally moved away to his throne without saying a word. But although Tom's ear tingled, his heart was jubilant.

As the school quieted down Tom made an honest effort to study, but the turmoil within him was too great. In turn he took his place in the reading class and made a botch of it; then in the geography class and turned lakes into mountains, mountains into rivers, and rivers into continents, till chaos was come again; then in the spelling class, and got "turned down," by a succession of mere baby words, till he brought up at the foot and yielded up the pewter medal which he had worn with ostentation for months.

The Adventures of Tom Sawyer/Chapter VII

The harder Tom tried to fasten his mind on his book, the more his ideas wandered. So at last, with a sigh and a yawn, he gave it up. It seemed to him that the noon recess would never come. The air was utterly dead. There was not a breath stirring. It was the sleepest of sleepy days. The drowsing murmur of the five and twenty studying scholars soothed the soul like the spell that is in the murmur of bees. Away off in the flaming sunshine, Cardiff Hill lifted its soft green sides through a shimmering veil of heat, tinted with the purple of distance; a few birds floated on lazy wing high in the air; no other living thing was visible but some cows, and they were asleep. Tom's heart ached to be free, or else to have something of interest to do to pass the dreary time. His hand wandered into his pocket and his face lit up with a glow of gratitude that was prayer, though he did not know it. Then furtively the percussion-cap box came out. He released the tick and put him on the long flat desk. The creature probably glowed with a gratitude that amounted to prayer, too, at this moment, but it was premature: for when he started thankfully to travel off, Tom turned him aside with a pin and made him take a new direction.

Tom's bosom friend sat next him, suffering just as Tom had been, and now he was deeply and gratefully interested in this entertainment in an instant. This bosom friend was Joe Harper. The two boys were sworn friends all the week, and embattled enemies on Saturdays. Joe took a pin out of his lapel and began to assist in exercising the prisoner. The sport grew in interest momentarily. Soon Tom said that they were interfering with each other, and neither getting the fullest benefit of the tick. So he put Joe's slate on the desk and drew a line down the middle of it from top to bottom.

"Now," said he, "as long as he is on your side you can stir him up and I'll let him alone; but if you let him get away and get on my side, you're to leave him alone as long as I can keep him from crossing over."

"All right, go ahead; start him up."

The tick escaped from Tom, presently, and crossed the equator. Joe harassed him awhile, and then he got away and crossed back again. This change of base occurred often. While one boy was worrying the tick with absorbing

interest, the other would look on with interest as strong, the two heads bowed together over the slate, and the two souls dead to all things else. At last luck seemed to settle and abide with Joe. The tick tried this, that, and the other course, and got as excited and as anxious as the boys themselves, but time and again just as he would have victory in his very grasp, so to speak, and Tom's fingers would be twitching to begin, Joe's pin would deftly head him off, and keep possession. At last Tom could stand it no longer. The temptation was too strong. So he reached out and lent a hand with his pin. Joe was angry in a moment. Said he:

"Tom, you let him alone."

"I only just want to stir him up a little, Joe."

"No, sir, it ain't fair; you just let him alone."

"Blame it, I ain't going to stir him much."

"Let him alone, I tell you."

"I won't!"

"You shall -- he's on my side of the line."

"Look here, Joe Harper, whose is that tick?"

"I don't care whose tick he is -- he's on my side of the line, and you sha'n't touch him."

"Well, I'll just bet I will, though. He's my tick and I'll do what I blame please with him, or die!"

A tremendous whack came down on Tom's shoulders, and its duplicate on Joe's; and for the space of two minutes the dust continued to fly from the two jackets and the whole school to enjoy it. The boys had been too absorbed to notice the hush that had stolen upon the school awhile before when the master came tiptoeing down the room and stood over them. He had contemplated a good part of the performance before he contributed his bit of variety to it.

When school broke up at noon, Tom flew to Becky Thatcher, and whispered in her ear:

"Put on your bonnet and let on you're going home; and when you get to the corner, give the rest of 'em the slip, and turn down through the lane and come back. I'll go the other way and come it over 'em the same way."

So the one went off with one group of scholars, and the other with another. In a little while the two met at the bottom of the lane, and when they reached the school they had it all to themselves. Then they sat together, with a slate before them, and Tom gave Becky the pencil and held her hand in his, guiding it, and so created another surprising house. When the interest in art began to wane, the two fell to talking. Tom was swimming in bliss. He said:

"Do you love rats?"

"No! I hate them!"

"Well, I do, too -- live ones. But I mean dead ones, to swing round your head with a string."

"No, I don't care for rats much, anyway. What I like is chewing-gum."

"Oh, I should say so! I wish I had some now."

"Do you? I've got some. I'll let you chew it awhile, but you must give it back to me."

That was agreeable, so they chewed it turn about, and dangled their legs against the bench in excess of contentment.

"Was you ever at a circus?" said Tom.

"Yes, and my pa's going to take me again some time, if I'm good."

"I been to the circus three or four times -- lots of times. Church ain't shucks to a circus. There's things going on at a circus all the time. I'm going to be a clown in a circus when I grow up."

"Oh, are you! That will be nice. They're so lovely, all spotted up."

"Yes, that's so. And they get slathers of money -- most a dollar a day, Ben Rogers says. Say, Becky, was you ever engaged?"

"What's that?"

"Why, engaged to be married."

"No."

"Would you like to?"

"I reckon so. I don't know. What is it like?"

"Like? Why it ain't like anything. You only just tell a boy you won't ever have anybody but him, ever ever ever, and then you kiss and that's all. Anybody can do it."

"Kiss? What do you kiss for?"

"Why, that, you know, is to -- well, they always do that."

"Everybody?"

"Why, yes, everybody that's in love with each other. Do you remember what I wrote on the slate?"

"Ye -- yes."

"What was it?"

"I sha'n't tell you."

"Shall I tell you?"

"Ye -- yes -- but some other time."

"No, now."

"No, not now -- to-morrow."

"Oh, no, now. Please, Becky -- I'll whisper it, I'll whisper it ever so easy."

Becky hesitating, Tom took silence for consent, and passed his arm about her waist and whispered the tale ever so softly, with his mouth close to her ear. And then he added:

"Now you whisper it to me -- just the same."

She resisted, for a while, and then said:

"You turn your face away so you can't see, and then I will. But you mustn't ever tell anybody -- Will you, Tom? Now you won't, will you?"

"No, indeed, indeed I won't. Now, Becky."

He turned his face away. She bent timidly around till her breath stirred his curls and whispered, "I -- love -- you!"

Then she sprang away and ran around and around the desks and benches, with Tom after her, and took refuge in a corner at last, with her little white apron to her face. Tom clasped her about her neck and pleaded:

"Now, Becky, it's all done -- all over but the kiss. Don't you be afraid of that -- it ain't anything at all. Please, Becky." And he tugged at her apron and the hands.

By and by she gave up, and let her hands drop; her face, all glowing with the struggle, came up and submitted. Tom kissed the red lips and said:

"Now it's all done, Becky. And always after this, you know, you ain't ever to love anybody but me, and you ain't ever to marry anybody but me, ever never and forever. Will you?"

"No, I'll never love anybody but you, Tom, and I'll never marry anybody but you -- and you ain't to ever marry anybody but me, either."

"Certainly. Of course. That's part of it. And always coming to school or when we're going home, you're to walk with me, when there ain't anybody looking -- and you choose me and I choose you at parties, because that's the way you do when you're engaged."

"It's so nice. I never heard of it before."

"Oh, it's ever so gay! Why, me and Amy Lawrence --"

The big eyes told Tom his blunder and he stopped, confused.

"Oh, Tom! Then I ain't the first you've ever been engaged to!"

The child began to cry. Tom said:

"Oh, don't cry, Becky, I don't care for her any more."

"Yes, you do, Tom -- you know you do."

Tom tried to put his arm about her neck, but she pushed him away and turned her face to the wall, and went on crying. Tom tried again, with soothing words in his mouth, and was repulsed again. Then his pride was up, and he strode away and went outside. He stood about, restless and uneasy, for a while, glancing at the door, every now and then, hoping she would repent and come to find him. But she did not. Then he began to feel badly and fear that he was in the wrong. It was a hard struggle with him to make new advances, now, but he nerved himself to it and entered. She was still standing back there in the corner, sobbing, with her face to the wall. Tom's heart smote him. He went to her and stood a moment, not knowing exactly how to proceed. Then he said hesitatingly:

"Becky, I -- I don't care for anybody but you."

No reply -- but sobs.

"Becky" -- pleadingly. "Becky, won't you say something?"

More sobs.

Tom got out his chiefest jewel, a brass knob from the top of an andiron, and passed it around her so that she could see it, and said:

"Please, Becky, won't you take it?"

She struck it to the floor. Then Tom marched out of the house and over the hills and far away, to return to school no more that day. Presently Becky began to suspect. She ran to the door; he was not in sight; she flew around to the play-yard; he was not there. Then she called:

"Tom! Come back, Tom!"

She listened intently, but there was no answer. She had no companions but silence and loneliness. So she sat down to cry again and upbraid herself; and by this time the scholars began to gather again, and she had to hide her griefs and still her broken heart and take up the cross of a long, dreary, aching afternoon, with none among the strangers about her to exchange sorrows with.

The Adventures of Tom Sawyer/Chapter VIII

Tom dodged hither and thither through lanes until he was well out of the track of returning scholars, and then fell into a moody jog. He crossed a small "branch" two or three times, because of a prevailing juvenile superstition that to cross water baffled pursuit. Half an hour later he was disappearing behind the Douglas mansion on the summit of Cardiff Hill, and the school-house was hardly distinguishable away off in the valley behind him. He entered a dense wood, picked his pathless way to the centre of it, and sat down on a mossy spot under a spreading oak. There was not even a zephyr stirring; the dead noonday heat had even stilled the songs of the birds; nature lay in a trance that was broken by no sound but the occasional far-off hammering of a woodpecker, and this seemed to render the pervading silence and sense of loneliness the more profound. The boy's soul was steeped in melancholy; his feelings were in happy accord with his surroundings. He sat long with his elbows on his knees and his chin in his hands, meditating. It seemed to him that life was but a trouble, at best, and he more than half envied Jimmy Hodges, so lately released; it must be very peaceful, he thought, to lie and slumber and dream forever and ever, with the wind whispering through the trees and caressing the grass and the flowers over the grave, and nothing to bother and grieve about, ever any more. If he only had a clean Sunday-school record he could be willing to go, and be done with it all. Now as to this girl. What had he done? Nothing. He had meant the best in the world, and been treated like a dog -- like a very dog. She would be sorry some day -- maybe when it was too late. Ah, if he could only die temporarily!

But the elastic heart of youth cannot be compressed into one constrained shape long at a time. Tom presently began to drift insensibly back into the concerns of this life again. What if he turned his back, now, and disappeared mysteriously? What if he went away -- ever so far away, into unknown countries beyond the seas -- and never came back any more! How would she feel then! The idea of being a clown recurred to him now, only to fill him with disgust. For frivolity and jokes and spotted tights were an offense, when they intruded themselves upon a spirit that was exalted into the vague august realm of the romantic. No, he would be a soldier, and return after long years, all war-worn and illustrious. No -- better still, he would join the Indians, and hunt buffaloes and go on the warpath in the mountain ranges and the trackless great plains of the Far West, and away in the future come back a great chief, bristling with feathers, hideous with paint, and prance into Sunday-school, some drowsy summer morning, with a blood-curdling war-whoop, and sear the eyeballs of all his companions with unappeasable envy. But no, there was something gaudier even than this. He would be a pirate! That was it! Now his future lay plain before him, and glowing with unimaginable splendor. How his name would fill the world, and make people shudder! How gloriously he would go plowing the dancing seas, in his long, low, black-hulled racer, the Spirit of the Storm, with his grisly flag flying at the fore! And at the zenith of his fame, how he would suddenly appear at the old village and stalk into church, brown and weather-beaten, in his black velvet doublet and trunks, his great jack-boots, his crimson sash, his belt bristling with horse-pistols, his crime-rusted cutlass at his side, his slouch hat with waving plumes, his black flag unfurled, with the skull and crossbones on it, and hear with swelling ecstasy the whisperings, "It's Tom Sawyer the Pirate! -- the Black Avenger of the Spanish Main!"

Yes, it was settled; his career was determined. He would run away from home and enter upon it. He would start the very next morning. Therefore he must now begin to get ready. He would collect his resources together. He went to a rotten log near at hand and began to dig under one end of it with his Barlow knife. He soon struck wood that sounded hollow. He put his hand there and uttered this incantation impressively:

"What hasn't come here, come! What's here, stay here!"

Then he scraped away the dirt, and exposed a pine shingle. He took it up and disclosed a shapely little treasure-house whose bottom and sides were of shingles. In it lay a marble. Tom's astonishment was boundless! He scratched his head with a perplexed air, and said:

"Well, that beats anything!"

Then he tossed the marble away pettishly, and stood cogitating. The truth was, that a superstition of his had failed, here, which he and all his comrades had always looked upon as infallible. If you buried a marble with certain necessary incantations, and left it alone a fortnight, and then opened the place with the incantation he had just used, you would find that all the marbles you had ever lost had gathered themselves together there, meantime, no matter how widely they had been separated. But now, this thing had actually and unquestionably failed. Tom's whole structure of faith was shaken to its foundations. He had many a time heard of this thing succeeding but never of its failing before. It did not occur to him that he had tried it several times before, himself, but could never find the hiding-places afterward. He puzzled over the matter some time, and finally decided that some witch had interfered and broken the charm. He thought he would satisfy himself on that point; so he searched around till he found a small sandy spot with a little funnel-shaped depression in it. He laid himself down and put his mouth close to this depression and called --

"Doodle-bug, doodle-bug, tell me what I want to know! Doodle-bug, doodle-bug, tell me what I want to know!"

The sand began to work, and presently a small black bug appeared for a second and then darted under again in a fright.

"He dasn't tell! So it was a witch that done it. I just knowed it."

He well knew the futility of trying to contend against witches, so he gave up discouraged. But it occurred to him that he might as well have the marble he had just thrown away, and therefore he went and made a patient search for it. But he could not find it. Now he went back to his treasure-house and carefully placed himself just as he had been standing when he tossed the marble away; then he took another marble from his pocket and tossed it in the same way, saying:

"Brother, go find your brother!"

He watched where it stopped, and went there and looked. But it must have fallen short or gone too far; so he tried twice more. The last repetition was successful. The two marbles lay within a foot of each other.

Just here the blast of a toy tin trumpet came faintly down the green aisles of the forest. Tom flung off his jacket and trousers, turned a suspender into a belt, raked away some brush behind the rotten log, disclosing a rude bow and arrow, a lath sword and a tin trumpet, and in a moment had seized these things and bounded away, barelegged, with fluttering shirt. He presently halted under a great elm, blew an answering blast, and then began to tiptoe and look warily out, this way and that. He said cautiously -- to an imaginary company:

"Hold, my merry men! Keep hid till I blow."

Now appeared Joe Harper, as airily clad and elaborately armed as Tom. Tom called:

"Hold! Who comes here into Sherwood Forest without my pass?"

"Guy of Guisborne wants no man's pass. Who art thou that -- that --"

"Dares to hold such language," said Tom, prompting -- for they talked "by the book," from memory.

"Who art thou that dares to hold such language?"

"I, indeed! I am Robin Hood, as thy caitiff carcase soon shall know."

"Then art thou indeed that famous outlaw? Right gladly will I dispute with thee the passes of the merry wood. Have at thee!"

They took their lath swords, dumped their other traps on the ground, struck a fencing attitude, foot to foot, and began a grave, careful combat, "two up and two down." Presently Tom said:

"Now, if you've got the hang, go it lively!"

So they "went it lively," panting and perspiring with the work. By and by Tom shouted:

"Fall! fall! Why don't you fall?"

"I sha'n't! Why don't you fall yourself? You're getting the worst of it."

"Why, that ain't anything. I can't fall; that ain't the way it is in the book. The book says, 'Then with one back-handed stroke he slew poor Guy of Guisborne.' You're to turn around and let me hit you in the back."

There was no getting around the authorities, so Joe turned, received the whack and fell.

"Now," said Joe, getting up, "you got to let me kill you. That's fair."

"Why, I can't do that, it ain't in the book."

"Well, it's blamed mean -- that's all."

"Well, say, Joe, you can be Friar Tuck or Much the miller's son, and lam me with a quarter-staff; or I'll be the Sheriff of Nottingham and you be Robin Hood a little while and kill me."

This was satisfactory, and so these adventures were carried out. Then Tom became Robin Hood again, and was allowed by the treacherous nun to bleed his strength away through his neglected wound. And at last Joe, representing a whole tribe of weeping outlaws, dragged him sadly forth, gave his bow into his feeble hands, and Tom said, "Where this arrow falls, there bury poor Robin Hood under the greenwood tree." Then he shot the arrow and fell back and would have died, but he lit on a nettle and sprang up too gaily for a corpse.

The boys dressed themselves, hid their accoutrements, and went off grieving that there were no outlaws any more, and wondering what modern civilization could claim to have done to compensate for their loss. They said they would rather be outlaws a year in Sherwood Forest than President of the United States forever.

The Adventures of Tom Sawyer/Chapter IX

At half-past nine, that night, Tom and Sid were sent to bed, as usual. They said their prayers, and Sid was soon asleep. Tom lay awake and waited, in restless impatience. When it seemed to him that it must be nearly daylight, he heard the clock strike ten! This was despair. He would have tossed and fidgeted, as his nerves demanded, but he was afraid he might wake Sid. So he lay still, and stared up into the dark. Everything was dismally still. By and by, out of the stillness, little, scarcely perceptible noises began to emphasize themselves. The ticking of the clock began to bring itself into notice. Old beams began to crack mysteriously. The stairs creaked faintly. Evidently spirits were abroad. A measured, muffled snore issued from Aunt Polly's chamber. And now the tiresome chirping of a cricket that no human ingenuity could locate, began. Next the ghastly ticking of a deathwatch in the wall at the bed's head made Tom shudder -- it meant that somebody's days were numbered. Then the howl of a far-off dog rose on the night air, and was answered by a fainter howl from a remoter distance. Tom was in an agony. At last he was satisfied that time had ceased and eternity begun; he began to doze, in spite of himself; the clock chimed eleven, but he did not hear it. And then there came, mingling with his half-formed dreams, a most melancholy caterwauling. The raising of a neighboring window disturbed him. A cry of "Scat! you devil!" and the crash of an empty bottle against the back of his aunt's woodshed brought him wide awake, and a single minute later he was dressed and out of the window and creeping along the roof of the "ell" on all fours. He "meow'd" with caution once or twice, as he went; then jumped to the roof of the woodshed and thence to the ground. Huckleberry Finn was there, with his dead cat. The boys moved off and disappeared in the gloom. At the end of half an hour they were wading through the tall grass of the graveyard.

It was a graveyard of the old-fashioned Western kind. It was on a hill, about a mile and a half from the village. It had a crazy board fence around it, which leaned inward in places, and outward the rest of the time, but stood upright nowhere. Grass and weeds grew rank over the whole cemetery. All the old graves were sunken in, there was not a tombstone on the place; round-topped, worm-eaten boards staggered over the graves, leaning for support and finding none. "Sacred to the memory of" So-and-So had been painted on them once, but it could no longer have been read, on the most of them, now, even if there had been light.

A faint wind moaned through the trees, and Tom feared it might be the spirits of the dead, complaining at being disturbed. The boys talked little, and only under their breath, for the time and the place and the pervading solemnity

and silence oppressed their spirits. They found the sharp new heap they were seeking, and ensconced themselves within the protection of three great elms that grew in a bunch within a few feet of the grave.

Then they waited in silence for what seemed a long time. The hooting of a distant owl was all the sound that troubled the dead stillness. Tom's reflections grew oppressive. He must force some talk. So he said in a whisper:

"Hucky, do you believe the dead people like it for us to be here?"

Huckleberry whispered:

"I wisht I knowed. It's awful solemn like, ain't it?"

"I bet it is."

There was a considerable pause, while the boys canvassed this matter inwardly. Then Tom whispered:

"Say, Hucky -- do you reckon Hoss Williams hears us talking?"

"O' course he does. Least his sperrit does."

Tom, after a pause:

"I wish I'd said Mister Williams. But I never meant any harm. Everybody calls him Hoss."

"A body can't be too partic'lar how they talk 'bout these-yer dead people, Tom."

This was a damper, and conversation died again.

Presently Tom seized his comrade's arm and said:

"Sh!"

"What is it, Tom?" And the two clung together with beating hearts.

"Sh! There 'tis again! Didn't you hear it?"

"I --"

"There! Now you hear it."

"Lord, Tom, they're coming! They're coming, sure. What'll we do?"

"I dono. Think they'll see us?"

"Oh, Tom, they can see in the dark, same as cats. I wisht I hadn't come."

"Oh, don't be afeard. I don't believe they'll bother us. We ain't doing any harm. If we keep perfectly still, maybe they won't notice us at all."

"I'll try to, Tom, but, Lord, I'm all of a shiver."

"Listen!"

The boys bent their heads together and scarcely breathed. A muffled sound of voices floated up from the far end of the graveyard.

"Look! See there!" whispered Tom. "What is it?"

"It's devil-fire. Oh, Tom, this is awful."

Some vague figures approached through the gloom, swinging an old-fashioned tin lantern that freckled the ground with innumerable little spangles of light. Presently Huckleberry whispered with a shudder:

"It's the devils sure enough. Three of 'em! Lordy, Tom, we're goners! Can you pray?"

"I'll try, but don't you be afeard. They ain't going to hurt us. 'Now I lay me down to sleep, I --"

"Sh!"

"What is it, Huck?"

"They're humans! One of 'em is, anyway. One of 'em's old Muff Potter's voice."

"No -- 'tain't so, is it?"

"I bet I know it. Don't you stir nor budge. He ain't sharp enough to notice us. Drunk, the same as usual, likely -- blamed old rip!"

"All right, I'll keep still. Now they're stuck. Can't find it. Here they come again. Now they're hot. Cold again. Hot again. Red hot! They're p'inted right, this time. Say, Huck, I know another o' them voices; it's Injun Joe."

"That's so -- that murderin' half-breed! I'd druther they was devils a dern sight. What kin they be up to?"

The whisper died wholly out, now, for the three men had reached the grave and stood within a few feet of the boys' hiding-place.

"Here it is," said the third voice; and the owner of it held the lantern up and revealed the face of young Doctor Robinson.

Potter and Injun Joe were carrying a handbarrow with a rope and a couple of shovels on it. They cast down their load and began to open the grave. The doctor put the lantern at the head of the grave and came and sat down with his back against one of the elm trees. He was so close the boys could have touched him.

"Hurry, men!" he said, in a low voice; "the moon might come out at any moment."

They growled a response and went on digging. For some time there was no noise but the grating sound of the spades discharging their freight of mould and gravel. It was very monotonous. Finally a spade struck upon the coffin with a dull woody accent, and within another minute or two the men had hoisted it out on the ground. They pried off the lid with their shovels, got out the body and dumped it rudely on the ground. The moon drifted from behind the clouds and exposed the pallid face. The barrow was got ready and the corpse placed on it, covered with a blanket, and bound to its place with the rope. Potter took out a large spring-knife and cut off the dangling end of the rope and then said:

"Now the cussed thing's ready, Sawbones, and you'll just out with another five, or here she stays."

"That's the talk!" said Injun Joe.

"Look here, what does this mean?" said the doctor. "You required your pay in advance, and I've paid you."

"Yes, and you done more than that," said Injun Joe, approaching the doctor, who was now standing. "Five years ago you drove me away from your father's kitchen one night, when I come to ask for something to eat, and you said I warn't there for any good; and when I swore I'd get even with you if it took a hundred years, your father had me jailed for a vagrant. Did you think I'd forget? The Injun blood ain't in me for nothing. And now I've got you, and you got to settle, you know!"

He was threatening the doctor, with his fist in his face, by this time. The doctor struck out suddenly and stretched the ruffian on the ground. Potter dropped his knife, and exclaimed:

"Here, now, don't you hit my pard!" and the next moment he had grappled with the doctor and the two were struggling with might and main, trampling the grass and tearing the ground with their heels. Injun Joe sprang to his feet, his eyes flaming with passion, snatched up Potter's knife, and went creeping, catlike and stooping, round and round about the combatants, seeking an opportunity. All at once the doctor flung himself free, seized the heavy headboard of Williams' grave and felled Potter to the earth with it -- and in the same instant the half-breed saw his chance and drove the knife to the hilt in the young man's breast. He reeled and fell partly upon Potter, flooding him with his blood, and in the same moment the clouds blotted out the dreadful spectacle and the two frightened boys went speeding away in the dark.

Presently, when the moon emerged again, Injun Joe was standing over the two forms, contemplating them. The doctor murmured inarticulately, gave a long gasp or two and was still. The half-breed muttered:

"That score is settled -- damn you."

Then he robbed the body. After which he put the fatal knife in Potter's open right hand, and sat down on the dismantled coffin. Three -- four -- five minutes passed, and then Potter began to stir and moan. His hand closed upon the knife; he raised it, glanced at it, and let it fall, with a shudder. Then he sat up, pushing the body from him, and

gazed at it, and then around him, confusedly. His eyes met Joe's.

"Lord, how is this, Joe?" he said.

"It's a dirty business," said Joe, without moving.

"What did you do it for?"

"I! I never done it!"

"Look here! That kind of talk won't wash."

Potter trembled and grew white.

"I thought I'd got sober. I'd no business to drink to-night. But it's in my head yet -- worse'n when we started here. I'm all in a muddle; can't recollect anything of it, hardly. Tell me, Joe -- honest, now, old feller -- did I do it? Joe, I never meant to -- 'pon my soul and honor, I never meant to, Joe. Tell me how it was, Joe. Oh, it's awful -- and him so young and promising."

"Why, you two was scuffling, and he fetched you one with the headboard and you fell flat; and then up you come, all reeling and staggering like, and snatched the knife and jammed it into him, just as he fetched you another awful clip -- and here you've laid, as dead as a wedge til now."

"Oh, I didn't know what I was a-doing. I wish I may die this minute if I did. It was all on account of the whiskey and the excitement, I reckon. I never used a weepion in my life before, Joe. I've fought, but never with weepions. They'll all say that. Joe, don't tell! Say you won't tell, Joe -- that's a good feller. I always liked you, Joe, and stood up for you, too. Don't you remember? You won't tell, will you, Joe?" And the poor creature dropped on his knees before the stolid murderer, and clasped his appealing hands.

"No, you've always been fair and square with me, Muff Potter, and I won't go back on you. There, now, that's as fair as a man can say."

"Oh, Joe, you're an angel. I'll bless you for this the longest day I live." And Potter began to cry.

"Come, now, that's enough of that. This ain't any time for blubbering. You be off yonder way and I'll go this. Move, now, and don't leave any tracks behind you."

Potter started on a trot that quickly increased to a run. The half-breed stood looking after him. He muttered:

"If he's as much stunned with the lick and fuddled with the rum as he had the look of being, he won't think of the knife till he's gone so far he'll be afraid to come back after it to such a place by himself -- chicken-heart!"

Two or three minutes later the murdered man, the blanketed corpse, the lidless coffin, and the open grave were under no inspection but the moon's. The stillness was complete again, too.

The Adventures of Tom Sawyer/Chapter X

The two boys flew on and on, toward the village, speechless with horror. They glanced backward over their shoulders from time to time, apprehensively, as if they feared they might be followed. Every stump that started up in their path seemed a man and an enemy, and made them catch their breath; and as they sped by some outlying cottages that lay near the village, the barking of the aroused watch-dogs seemed to give wings to their feet.

"If we can only get to the old tannery before we break down!" whispered Tom, in short catches between breaths. "I can't stand it much longer."

Huckleberry's hard pantings were his only reply, and the boys fixed their eyes on the goal of their hopes and bent to their work to win it. They gained steadily on it, and at last, breast to breast, they burst through the open door and fell grateful and exhausted in the sheltering shadows beyond. By and by their pulses slowed down, and Tom whispered:

"Huckleberry, what do you reckon'll come of this?"

"If Doctor Robinson dies, I reckon hanging'll come of it."

"Do you though?"

"Why, I know it, Tom."

Tom thought a while, then he said:

"Who'll tell? We?"

"What are you talking about? S'pose something happened and Injun Joe didn't hang? Why, he'd kill us some time or other, just as dead sure as we're a laying here."

"That's just what I was thinking to myself, Huck."

"If anybody tells, let Muff Potter do it, if he's fool enough. He's generally drunk enough."

Tom said nothing -- went on thinking. Presently he whispered:

"Huck, Muff Potter don't know it. How can he tell?"

"What's the reason he don't know it?"

"Because he'd just got that whack when Injun Joe done it. D'you reckon he could see anything? D'you reckon he knowed anything?"

"By hokey, that's so, Tom!"

"And besides, look-a-here -- maybe that whack done for him!"

"No, 'taint likely, Tom. He had liquor in him; I could see that; and besides, he always has. Well, when pap's full, you might take and belt him over the head with a church and you couldn't phase him. He says so, his own self. So it's the same with Muff Potter, of course. But if a man was dead sober, I reckon maybe that whack might fetch him; I dono."

After another reflective silence, Tom said:

"Hucky, you sure you can keep mum?"

"Tom, we got to keep mum. You know that. That Injun devil wouldn't make any more of drownding us than a couple of cats, if we was to squeak 'bout this and they didn't hang him. Now, look-a-here, Tom, less take and swear to one another -- that's what we got to do -- swear to keep mum."

"I'm agreed. It's the best thing. Would you just hold hands and swear that we --"

"Oh no, that wouldn't do for this. That's good enough for little rubbishy common things -- specially with gals, cuz they go back on you anyway, and blab if they get in a huff -- but there orter be writing 'bout a big thing like this. And blood."

Tom's whole being applauded this idea. It was deep, and dark, and awful; the hour, the circumstances, the surroundings, were in keeping with it. He picked up a clean pine shingle that lay in the moonlight, took a little

fragment of "red keel" out of his pocket, got the moon on his work, and painfully scrawled these lines, emphasizing each slow down-stroke by clamping his tongue between his teeth, and letting up the pressure on the up-strokes.

```
"Huck Finn and  
Tom Sawyer swears  
they will keep mum  
about this and they  
wish they may drop  
down dead in their  
tracks if they ever  
tell and rot.
```

Huckleberry was filled with admiration of Tom's facility in writing, and the sublimity of his language. He at once took a pin from his lapel and was going to prick his flesh, but Tom said: "Hold on! Don't do that. A pin's brass. It might have verdigrease on it."

"What's verdigrease?"

"It's p'ison. That's what it is. You just swallow some of it once -- you'll see."

So Tom unwound the thread from one of his needles, and each boy pricked the ball of his thumb and squeezed out a drop of blood. In time, after many squeezes, Tom managed to sign his initials, using the ball of his little finger for a pen. Then he showed Huckleberry how to make an H and an F, and the oath was complete. They buried the shingle close to the wall, with some dismal ceremonies and incantations, and the fetters that bound their tongues were considered to be locked and the key thrown away.

A figure crept stealthily through a break in the other end of the ruined building, now, but they did not notice it.

"Tom," whispered Huckleberry, "does this keep us from ever telling -- always?"

"Of course it does. It don't make any difference what happens, we got to keep mum. We'd drop down dead -- don't you know that?"

"Yes, I reckon that's so."

They continued to whisper for some little time. Presently a dog set up a long, lugubrious howl just outside -- within ten feet of them. The boys clasped each other suddenly, in an agony of fright.

"Which of us does he mean?" gasped Huckleberry.

"I dono -- peep through the crack. Quick!"

"No, you, Tom!"

"I can't -- I can't do it, Huck!"

"Please, Tom. There 'tis again!"

"Oh, lordy, I'm thankful!" whispered Tom. "I know his voice. It's Bull Harbison." *

[* If Mr. Harbison owned a slave named Bull, Tom would have spoken of him as "Harbison's Bull," but a son or a dog of that name was "Bull Harbison."]

"Oh, that's good -- I tell you, Tom, I was most scared to death; I'd a bet anything it was a stray dog."

The dog howled again. The boys' hearts sank once more.

"Oh, my! that ain't no Bull Harbison!" whispered Huckleberry. "Do, Tom!"

Tom, quaking with fear, yielded, and put his eye to the crack. His whisper was hardly audible when he said:

"Oh, Huck, it's a stray dog!"

"Quick, Tom, quick! Who does he mean?"

"Huck, he must mean us both -- we're right together."

"Oh, Tom, I reckon we're goners. I reckon there ain't no mistake 'bout where I'll go to. I been so wicked."

"Dad fetch it! This comes of playing hookey and doing everything a feller's told not to do. I might a been good, like Sid, if I'd a tried -- but no, I wouldn't, of course. But if ever I get off this time, I lay I'll just waller in Sunday-schools!" And Tom began to snuffle a little.

"You bad!" and Huckleberry began to snuffle too. "Consound it, Tom Sawyer, you're just old pie, 'longside o' what I am. Oh, lordy, lordy, lordy, I wisht I only had half your chance."

Tom choked off and whispered:

"Look, Hucky, look! He's got his back to us!"

Hucky looked, with joy in his heart.

"Well, he has, by jingoes! Did he before?"

"Yes, he did. But I, like a fool, never thought. Oh, this is bully, you know. Now who can he mean?"

The howling stopped. Tom pricked up his ears.

"Sh! What's that?" he whispered.

"Sounds like -- like hogs grunting. No -- it's somebody snoring, Tom."

"That is it! Where 'bouts is it, Huck?"

"I bleeve it's down at 'tother end. Sounds so, anyway. Pap used to sleep there, sometimes, 'long with the hogs, but laws bless you, he just lifts things when he snores. Besides, I reckon he ain't ever coming back to this town any more."

The spirit of adventure rose in the boys' souls once more.

"Hucky, do you das't to go if I lead?"

"I don't like to, much. Tom, s'pose it's Injun Joe!"

Tom quailed. But presently the temptation rose up strong again and the boys agreed to try, with the understanding that they would take to their heels if the snoring stopped. So they went tiptoeing stealthily down, the one behind the other. When they had got to within five steps of the snorer, Tom stepped on a stick, and it broke with a sharp snap. The man moaned, writhed a little, and his face came into the moonlight. It was Muff Potter. The boys' hearts had stood still, and their hopes too, when the man moved, but their fears passed away now. They tiptoed out, through the broken weather-boarding, and stopped at a little distance to exchange a parting word. That long, lugubrious howl rose on the night air again! They turned and saw the strange dog standing within a few feet of where Potter was lying, and facing Potter, with his nose pointing heavenward.

"Oh, geeminy, it's him!" exclaimed both boys, in a breath.

"Say, Tom -- they say a stray dog come howling around Johnny Miller's house, 'bout midnight, as much as two weeks ago; and a whippoorwill come in and lit on the banisters and sung, the very same evening; and there ain't anybody dead there yet."

"Well, I know that. And suppose there ain't. Didn't Gracie Miller fall in the kitchen fire and burn herself terrible the very next Saturday?"

"Yes, but she ain't dead. And what's more, she's getting better, too."

"All right, you wait and see. She's a goner, just as dead sure as Muff Potter's a goner. That's what the niggers say, and they know all about these kind of things, Huck."

Then they separated, cogitating. When Tom crept in at his bedroom window the night was almost spent. He undressed with excessive caution, and fell asleep congratulating himself that nobody knew of his escapade. He was not aware that the gently-snoring Sid was awake, and had been so for an hour.

When Tom awoke, Sid was dressed and gone. There was a late look in the light, a late sense in the atmosphere. He was startled. Why had he not been called -- persecuted till he was up, as usual? The thought filled him with bodings.

Within five minutes he was dressed and down-stairs, feeling sore and drowsy. The family were still at table, but they had finished breakfast. There was no voice of rebuke; but there were averted eyes; there was a silence and an air of solemnity that struck a chill to the culprit's heart. He sat down and tried to seem gay, but it was up-hill work; it roused no smile, no response, and he lapsed into silence and let his heart sink down to the depths.

After breakfast his aunt took him aside, and Tom almost brightened in the hope that he was going to be flogged; but it was not so. His aunt wept over him and asked him how he could go and break her old heart so; and finally told him to go on, and ruin himself and bring her gray hairs with sorrow to the grave, for it was no use for her to try any more. This was worse than a thousand whippings, and Tom's heart was sorer now than his body. He cried, he pleaded for forgiveness, promised to reform over and over again, and then received his dismissal, feeling that he had won but an imperfect forgiveness and established but a feeble confidence.

He left the presence too miserable to even feel revengeful toward Sid; and so the latter's prompt retreat through the back gate was unnecessary. He moped to school gloomy and sad, and took his flogging, along with Joe Harper, for playing hookey the day before, with the air of one whose heart was busy with heavier woes and wholly dead to trifles. Then he betook himself to his seat, rested his elbows on his desk and his jaws in his hands, and stared at the wall with the stony stare of suffering that has reached the limit and can no further go. His elbow was pressing against some hard substance. After a long time he slowly and sadly changed his position, and took up this object with a sigh. It was in a paper. He unrolled it. A long, lingering, colossal sigh followed, and his heart broke. It was his brass andiron knob!

This final feather broke the camel's back.

The Adventures of Tom Sawyer/Chapter XI

Close upon the hour of noon the whole village was suddenly electrified with the ghastly news. No need of the as yet undreamed-of telegraph; the tale flew from man to man, from group to group, from house to house, with little less than telegraphic speed. Of course the schoolmaster gave holiday for that afternoon; the town would have thought strangely of him if he had not.

A gory knife had been found close to the murdered man, and it had been recognized by somebody as belonging to Muff Potter -- so the story ran. And it was said that a belated citizen had come upon Potter washing himself in the "branch" about one or two o'clock in the morning, and that Potter had at once sneaked off -- suspicious circumstances, especially the washing which was not a habit with Potter. It was also said that the town had been ransacked for this "murderer" (the public are not slow in the matter of sifting evidence and arriving at a verdict), but that he could not be found. Horsemen had departed down all the roads in every direction, and the Sheriff "was confident" that he would be captured before night.

All the town was drifting toward the graveyard. Tom's heartbreak vanished and he joined the procession, not because he would not a thousand times rather go anywhere else, but because an awful, unaccountable fascination drew him on. Arrived at the dreadful place, he wormed his small body through the crowd and saw the dismal spectacle. It seemed to him an age since he was there before. Somebody pinched his arm. He turned, and his eyes met Huckleberry's. Then both looked elsewhere at once, and wondered if anybody had noticed anything in their mutual glance. But everybody was talking, and intent upon the grisly spectacle before them.

"Poor fellow!" "Poor young fellow!" "This ought to be a lesson to grave robbers!" "Muff Potter'll hang for this if they catch him!" This was the drift of remark; and the minister said, "It was a judgment; His hand is here."

Now Tom shivered from head to heel; for his eye fell upon the stolid face of Injun Joe. At this moment the crowd began to sway and struggle, and voices shouted, "It's him! it's him! he's coming himself!"

"Who? Who?" from twenty voices.

"Muff Potter!"

"Hallo, he's stopped! -- Look out, he's turning! Don't let him get away!"

People in the branches of the trees over Tom's head said he wasn't trying to get away -- he only looked doubtful and perplexed.

"Infernal impudence!" said a bystander; "wanted to come and take a quiet look at his work, I reckon -- didn't expect any company."

The crowd fell apart, now, and the Sheriff came through, ostentatiously leading Potter by the arm. The poor fellow's face was haggard, and his eyes showed the fear that was upon him. When he stood before the murdered man, he shook as with a palsy, and he put his face in his hands and burst into tears.

"I didn't do it, friends," he sobbed; "pon my word and honor I never done it."

"Who's accused you?" shouted a voice.

This shot seemed to carry home. Potter lifted his face and looked around him with a pathetic hopelessness in his eyes. He saw Injun Joe, and exclaimed:

"Oh, Injun Joe, you promised me you'd never --"

"Is that your knife?" and it was thrust before him by the Sheriff.

Potter would have fallen if they had not caught him and eased him to the ground. Then he said:

"Something told me 't if I didn't come back and get --" He shuddered; then waved his nerveless hand with a vanquished gesture and said, "Tell 'em, Joe, tell 'em -- it ain't any use any more."

Then Huckleberry and Tom stood dumb and staring, and heard the stony-hearted liar reel off his serene statement, they expecting every moment that the clear sky would deliver God's lightnings upon his head, and wondering to see how long the stroke was delayed. And when he had finished and still stood alive and whole, their wavering impulse to break their oath and save the poor betrayed prisoner's life faded and vanished away, for plainly this miscreant had sold himself to Satan and it would be fatal to meddle with the property of such a power as that.

"Why didn't you leave? What did you want to come here for?" somebody said.

"I couldn't help it -- I couldn't help it," Potter moaned. "I wanted to run away, but I couldn't seem to come anywhere but here." And he fell to sobbing again.

Injun Joe repeated his statement, just as calmly, a few minutes afterward on the inquest, under oath; and the boys, seeing that the lightnings were still withheld, were confirmed in their belief that Joe had sold himself to the devil. He was now become, to them, the most balefully interesting object they had ever looked upon, and they could not take their fascinated eyes from his face.

They inwardly resolved to watch him nights, when opportunity should offer, in the hope of getting a glimpse of his dread master.

Injun Joe helped to raise the body of the murdered man and put it in a wagon for removal; and it was whispered through the shuddering crowd that the wound bled a little! The boys thought that this happy circumstance would turn suspicion in the right direction; but they were disappointed, for more than one villager remarked:

"It was within three feet of Muff Potter when it done it."

Tom's fearful secret and gnawing conscience disturbed his sleep for as much as a week after this; and at breakfast one morning Sid said:

"Tom, you pitch around and talk in your sleep so much that you keep me awake half the time."

Tom blanched and dropped his eyes.

"It's a bad sign," said Aunt Polly, gravely. "What you got on your mind, Tom?"

"Nothing. Nothing 't I know of." But the boy's hand shook so that he spilled his coffee.

"And you do talk such stuff," Sid said. "Last night you said, 'It's blood, it's blood, that's what it is!' You said that over and over. And you said, 'Don't torment me so -- I'll tell!' Tell what? What is it you'll tell?"

Everything was swimming before Tom. There is no telling what might have happened, now, but luckily the concern passed out of Aunt Polly's face and she came to Tom's relief without knowing it. She said:

"Sho! It's that dreadful murder. I dream about it most every night myself. Sometimes I dream it's me that done it."

Mary said she had been affected much the same way. Sid seemed satisfied. Tom got out of the presence as quick as he plausibly could, and after that he complained of toothache for a week, and tied up his jaws every night. He never knew that Sid lay nightly watching, and frequently slipped the bandage free and then leaned on his elbow listening a good while at a time, and afterward slipped the bandage back to its place again. Tom's distress of mind wore off gradually and the toothache grew irksome and was discarded. If Sid really managed to make anything out of Tom's disjointed mutterings, he kept it to himself.

It seemed to Tom that his schoolmates never would get done holding inquests on dead cats, and thus keeping his trouble present to his mind. Sid noticed that Tom never was coroner at one of these inquiries, though it had been his habit to take the lead in all new enterprises; he noticed, too, that Tom never acted as a witness -- and that was strange; and Sid did not overlook the fact that Tom even showed a marked aversion to these inquests, and always avoided them when he could. Sid marvelled, but said nothing. However, even inquests went out of vogue at last, and ceased to torture Tom's conscience.

Every day or two, during this time of sorrow, Tom watched his opportunity and went to the little grated jail-window and smuggled such small comforts through to the "murderer" as he could get hold of. The jail was a trifling little brick den that stood in a marsh at the edge of the village, and no guards were afforded for it; indeed, it was seldom occupied. These offerings greatly helped to ease Tom's conscience.

The villagers had a strong desire to tar-and-feather Injun Joe and ride him on a rail, for body-snatching, but so formidable was his character that nobody could be found who was willing to take the lead in the matter, so it was dropped. He had been careful to begin both of his inquest-statements with the fight, without confessing the grave-robbery that preceded it; therefore it was deemed wisest not to try the case in the courts at present.

The Adventures of Tom Sawyer/Chapter XII

One of the reasons why Tom's mind had drifted away from its secret troubles was, that it had found a new and weighty matter to interest itself about. Becky Thatcher had stopped coming to school. Tom had struggled with his pride a few days, and tried to "whistle her down the wind," but failed. He began to find himself hanging around her father's house, nights, and feeling very miserable. She was ill. What if she should die! There was distraction in the thought. He no longer took an interest in war, nor even in piracy. The charm of life was gone; there was nothing but dreariness left. He put his hoop away, and his bat; there was no joy in them any more. His aunt was concerned. She began to try all manner of remedies on him. She was one of those people who are infatuated with patent medicines and all new-fangled methods of producing health or mending it. She was an inveterate experimenter in these things. When something fresh in this line came out she was in a fever, right away, to try it; not on herself, for she was never ailing, but on anybody else that came handy. She was a subscriber for all the "Health" periodicals and phrenological frauds; and the solemn ignorance they were inflated with was breath to her nostrils. All the "rot" they contained about ventilation, and how to go to bed, and how to get up, and what to eat, and what to drink, and how much exercise to take, and what frame of mind to keep one's self in, and what sort of clothing to wear, was all gospel to her, and she never observed that her health-journals of the current month customarily upset everything they had recommended the month before. She was as simple-hearted and honest as the day was long, and so she was an easy victim. She gathered together her quack periodicals and her quack medicines, and thus armed with death, went about on her pale horse, metaphorically speaking, with "hell following after." But she never suspected that she was not an angel of healing and the balm of Gilead in disguise, to the suffering neighbors. The water treatment was new, now, and Tom's low condition was a windfall to her. She had him out at daylight every morning, stood him up in the woodshed and drowned him with a deluge of cold water; then she scrubbed him down with a towel like a file, and so brought him to; then she rolled him up in a wet sheet and put him away under blankets till she sweated his soul clean and "the yellow stains of it came through his pores" -- as Tom said.

Yet notwithstanding all this, the boy grew more and more melancholy and pale and dejected. She added hot baths, sitz baths, shower baths, and plunges. The boy remained as dismal as a hearse. She began to assist the water with a slim oatmeal diet and blister-plasters. She calculated his capacity as she would a jug's, and filled him up every day with quack cure-alls.

Tom had become indifferent to persecution by this time. This phase filled the old lady's heart with consternation. This indifference must be broken up at any cost. Now she heard of Pain-killer for the first time. She ordered a lot at once. She tasted it and was filled with gratitude. It was simply fire in a liquid form. She dropped the water treatment and everything else, and pinned her faith to Pain-killer. She gave Tom a teaspoonful and watched with the deepest anxiety for the result. Her troubles were instantly at rest, her soul at peace again; for the "indifference" was broken up. The boy could not have shown a wilder, heartier interest, if she had built a fire under him.

Tom felt that it was time to wake up; this sort of life might be romantic enough, in his blighted condition, but it was getting to have too little sentiment and too much distracting variety about it. So he thought over various plans for relief, and finally hit pon that of professing to be fond of Pain-killer. He asked for it so often that he became a nuisance, and his aunt ended by telling him to help himself and quit bothering her. If it had been Sid, she would have had no misgivings to alloy her delight; but since it was Tom, she watched the bottle clandestinely. She found that the medicine did really diminish, but it did not occur to her that the boy was mending the health of a crack in the sitting-room floor with it.

One day Tom was in the act of dosing the crack when his aunt's yellow cat came along, purring, eying the teaspoon avariciously, and begging for a taste. Tom said:

"Don't ask for it unless you want it, Peter."

But Peter signified that he did want it.

"You better make sure."

Peter was sure.

"Now you've asked for it, and I'll give it to you, because there ain't anything mean about me; but if you find you don't like it, you mustn't blame anybody but your own self."

Peter was agreeable. So Tom pried his mouth open and poured down the Pain-killer. Peter sprang a couple of yards in the air, and then delivered a war-whoop and set off round and round the room, banging against furniture, upsetting flower-pots, and making general havoc. Next he rose on his hind feet and pranced around, in a frenzy of enjoyment, with his head over his shoulder and his voice proclaiming his unappeasable happiness. Then he went tearing around the house again spreading chaos and destruction in his path. Aunt Polly entered in time to see him throw a few double summersets, deliver a final mighty hurrah, and sail through the open window, carrying the rest of the flower-pots with him. The old lady stood petrified with astonishment, peering over her glasses; Tom lay on the floor expiring with laughter.

"Tom, what on earth ails that cat?"

"I don't know, aunt," gasped the boy.

"Why, I never see anything like it. What did make him act so?"

"Deed I don't know, Aunt Polly; cats always act so when they're having a good time."

"They do, do they?" There was something in the tone that made Tom apprehensive.

"Yes'm. That is, I believe they do."

"You do?"

"Yes'm."

The old lady was bending down, Tom watching, with interest emphasized by anxiety. Too late he divined her "drift." The handle of the telltale teaspoon was visible under the bed-valance. Aunt Polly took it, held it up. Tom winced, and dropped his eyes. Aunt Polly raised him by the usual handle -- his ear -- and cracked his head soundly with her thimble.

"Now, sir, what did you want to treat that poor dumb beast so, for?"

"I done it out of pity for him -- because he hadn't any aunt."

"Hadn't any aunt! -- you numskull. What has that got to do with it?"

"Heaps. Because if he'd had one she'd a burnt him out herself! She'd a roasted his bowels out of him 'thout any more feeling than if he was a human!"

Aunt Polly felt a sudden pang of remorse. This was putting the thing in a new light; what was cruelty to a cat might be cruelty to a boy, too. She began to soften; she felt sorry. Her eyes watered a little, and she put her hand on Tom's head and said gently:

"I was meaning for the best, Tom. And, Tom, it did do you good."

Tom looked up in her face with just a perceptible twinkle peeping through his gravity.

"I know you was meaning for the best, aunty, and so was I with Peter. It done him good, too. I never see him get around so since --"

"Oh, go 'long with you, Tom, before you aggravate me again. And you try and see if you can't be a good boy, for once, and you needn't take any more medicine."

Tom reached school ahead of time. It was noticed that this strange thing had been occurring every day latterly. And now, as usual of late, he hung about the gate of the schoolyard instead of playing with his comrades. He was sick, he said, and he looked it. He tried to seem to be looking everywhere but whither he really was looking -- down the road. Presently Jeff Thatcher hove in sight, and Tom's face lighted; he gazed a moment, and then turned sorrowfully away. When Jeff arrived, Tom accosted him; and "led up" warily to opportunities for remark about Becky, but the giddy

lad never could see the bait. Tom watched and watched, hoping whenever a frisking frock came in sight, and hating the owner of it as soon as he saw she was not the right one. At last frocks ceased to appear, and he dropped hopelessly into the dumps; he entered the empty schoolhouse and sat down to suffer. Then one more frock passed in at the gate, and Tom's heart gave a great bound. The next instant he was out, and "going on" like an Indian; yelling, laughing, chasing boys, jumping over the fence at risk of life and limb, throwing handsprings, standing on his head -- doing all the heroic things he could conceive of, and keeping a furtive eye out, all the while, to see if Becky Thatcher was noticing. But she seemed to be unconscious of it all; she never looked. Could it be possible that she was not aware that he was there? He carried his exploits to her immediate vicinity; came war-whooping around, snatched a boy's cap, hurled it to the roof of the schoolhouse, broke through a group of boys, tumbling them in every direction, and fell sprawling, himself, under Becky's nose, almost upsetting her -- and she turned, with her nose in the air, and he heard her say: "Mf! some people think they're mighty smart -- always showing off!"

Tom's cheeks burned. He gathered himself up and sneaked off, crushed and crestfallen.

The Adventures of Tom Sawyer/Chapter XIII

Tom's mind was made up now. He was gloomy and desperate. He was a forsaken, friendless boy, he said; nobody loved him; when they found out what they had driven him to, perhaps they would be sorry; he had tried to do right and get along, but they would not let him; since nothing would do them but to be rid of him, let it be so; and let them blame him for the consequences -- why shouldn't they? What right had the friendless to complain? Yes, they had forced him to it at last: he would lead a life of crime. There was no choice.

By this time he was far down Meadow Lane, and the bell for school to "take up" tinkled faintly upon his ear. He sobbed, now, to think he should never, never hear that old familiar sound any more -- it was very hard, but it was forced on him; since he was driven out into the cold world, he must submit -- but he forgave them. Then the sobs came thick and fast.

Just at this point he met his soul's sworn comrade, Joe Harper -- hard-eyed, and with evidently a great and dismal purpose in his heart. Plainly here were "two souls with but a single thought." Tom, wiping his eyes with his sleeve, began to blubber out something about a resolution to escape from hard usage and lack of sympathy at home by roaming abroad into the great world never to return; and ended by hoping that Joe would not forget him.

But it transpired that this was a request which Joe had just been going to make of Tom, and had come to hunt him up for that purpose. His mother had whipped him for drinking some cream which he had never tasted and knew nothing about; it was plain that she was tired of him and wished him to go; if she felt that way, there was nothing for him to do but succumb; he hoped she would be happy, and never regret having driven her poor boy out into the unfeeling world to suffer and die.

As the two boys walked sorrowing along, they made a new compact to stand by each other and be brothers and never separate till death relieved them of their troubles. Then they began to lay their plans. Joe was for being a hermit, and living on crusts in a remote cave, and dying, some time, of cold and want and grief; but after listening to Tom, he conceded that there were some conspicuous advantages about a life of crime, and so he consented to be a pirate.

Three miles below St. Petersburg, at a point where the Mississippi River was a trifle over a mile wide, there was a long, narrow, wooded island, with a shallow bar at the head of it, and this offered well as a rendezvous. It was not inhabited; it lay far over toward the further shore, abreast a dense and almost wholly unpeopled forest. So Jackson's Island was chosen. Who were to be the subjects of their piracies was a matter that did not occur to them. Then they hunted up Huckleberry Finn, and he joined them promptly, for all careers were one to him; he was indifferent. They presently separated to meet at a lonely spot on the river-bank two miles above the village at the favorite hour -- which was midnight. There was a small log raft there which they meant to capture. Each would bring hooks and lines, and such provision as he could steal in the most dark and mysterious way -- as became outlaws. And before the

afternoon was done, they had all managed to enjoy the sweet glory of spreading the fact that pretty soon the town would "hear something." All who got this vague hint were cautioned to "be mum and wait."

About midnight Tom arrived with a boiled ham and a few trifles, and stopped in a dense undergrowth on a small bluff overlooking the meeting-place. It was starlight, and very still. The mighty river lay like an ocean at rest. Tom listened a moment, but no sound disturbed the quiet. Then he gave a low, distinct whistle. It was answered from under the bluff. Tom whistled twice more; these signals were answered in the same way. Then a guarded voice said:

"Who goes there?"

"Tom Sawyer, the Black Avenger of the Spanish Main. Name your names."

"Huck Finn the Red-Handed, and Joe Harper the Terror of the Seas." Tom had furnished these titles, from his favorite literature.

"'Tis well. Give the countersign."

Two hoarse whispers delivered the same awful word simultaneously to the brooding night:

"Blood!"

Then Tom tumbled his ham over the bluff and let himself down after it, tearing both skin and clothes to some extent in the effort. There was an easy, comfortable path along the shore under the bluff, but it lacked the advantages of difficulty and danger so valued by a pirate.

The Terror of the Seas had brought a side of bacon, and had about worn himself out with getting it there. Finn the Red-Handed had stolen a skillet and a quantity of half-cured leaf tobacco, and had also brought a few corn-cobs to make pipes with. But none of the pirates smoked or "chewed" but himself. The Black Avenger of the Spanish Main said it would never do to start without some fire. That was a wise thought; matches were hardly known there in that day. They saw a fire smouldering upon a great raft a hundred yards above, and they went stealthily thither and helped themselves to a chunk. They made an imposing adventure of it, saying, "Hist!" every now and then, and suddenly halting with finger on lip; moving with hands on imaginary dagger-hilts; and giving orders in dismal whispers that if "the foe" stirred, to "let him have it to the hilt," because "dead men tell no tales." They knew well enough that the raftsmen were all down at the village laying in stores or having a spree, but still that was no excuse for their conducting this thing in an unpiratical way.

They shoved off, presently, Tom in command, Huck at the after oar and Joe at the forward. Tom stood amidships, gloomy-browed, and with folded arms, and gave his orders in a low, stern whisper:

"Luff, and bring her to the wind!"

"Aye-aye, sir!"

"Steady, steady-y-y-y!"

"Steady it is, sir!"

"Let her go off a point!"

"Point it is, sir!"

As the boys steadily and monotonously drove the raft toward mid-stream it was no doubt understood that these orders were given only for "style," and were not intended to mean anything in particular.

"What sail's she carrying?"

"Courses, tops'ls, and flying-jib, sir."

"Send the r'yals up! Lay out aloft, there, half a dozen of ye -- foretopmaststuns'! Lively, now!"

"Aye-aye, sir!"

"Shake out that maintogalans'! Sheets and braces! Now my hearties!"

"Aye-aye, sir!"

"Hellum-a-lee -- hard a port! Stand by to meet her when she comes! Port, port! Now, men! With a will! Stead-y-y-y!"

"Steady it is, sir!"

The raft drew beyond the middle of the river; the boys pointed her head right, and then lay on their oars. The river was not high, so there was not more than a two or three mile current. Hardly a word was said during the next three-quarters of an hour. Now the raft was passing before the distant town. Two or three glimmering lights showed where it lay, peacefully sleeping, beyond the vague vast sweep of star-gemmed water, unconscious of the tremendous event that was happening. The Black Avenger stood still with folded arms, "looking his last" upon the scene of his former joys and his later sufferings, and wishing "she" could see him now, abroad on the wild sea, facing peril and death with dauntless heart, going to his doom with a grim smile on his lips. It was but a small strain on his imagination to remove Jackson's Island beyond eyeshot of the village, and so he "looked his last" with a broken and satisfied heart. The other pirates were looking their last, too; and they all looked so long that they came near letting the current drift them out of the range of the island. But they discovered the danger in time, and made shift to avert it. About two o'clock in the morning the raft grounded on the bar two hundred yards above the head of the island, and they waded back and forth until they had landed their freight. Part of the little raft's belongings consisted of an old sail, and this they spread over a nook in the bushes for a tent to shelter their provisions; but they themselves would sleep in the open air in good weather, as became outlaws.

They built a fire against the side of a great log twenty or thirty steps within the sombre depths of the forest, and then cooked some bacon in the frying-pan for supper, and used up half of the corn "pone" stock they had brought. It seemed glorious sport to be feasting in that wild, free way in the virgin forest of an unexplored and uninhabited island, far from the haunts of men, and they said they never would return to civilization. The climbing fire lit up their faces and threw its ruddy glare upon the pillared tree-trunks of their forest temple, and upon the varnished foliage and festooning vines.

When the last crisp slice of bacon was gone, and the last allowance of corn pone devoured, the boys stretched themselves out on the grass, filled with contentment. They could have found a cooler place, but they would not deny themselves such a romantic feature as the roasting camp-fire.

"Ain't it gay?" said Joe.

"It's nuts!" said Tom. "What would the boys say if they could see us?"

"Say? Well, they'd just die to be here -- hey, Hucky!"

"I reckon so," said Huckleberry; "anyways, I'm suited. I don't want nothing better'n this. I don't ever get enough to eat, gen'ally -- and here they can't come and pick at a feller and bullyrag him so."

"It's just the life for me," said Tom. "You don't have to get up, mornings, and you don't have to go to school, and wash, and all that blame foolishness. You see a pirate don't have to do anything, Joe, when he's ashore, but a hermit he has to be praying considerable, and then he don't have any fun, anyway, all by himself that way."

"Oh yes, that's so," said Joe, "but I hadn't thought much about it, you know. I'd a good deal rather be a pirate, now that I've tried it."

"You see," said Tom, "people don't go much on hermits, nowadays, like they used to in old times, but a pirate's always respected. And a hermit's got to sleep on the hardest place he can find, and put sackcloth and ashes on his head, and stand out in the rain, and --"

"What does he put sackcloth and ashes on his head for?" inquired Huck.

"I dono. But they've got to do it. Hermits always do. You'd have to do that if you was a hermit."

"Dern'd if I would," said Huck.

"Well, what would you do?"

"I dono. But I wouldn't do that."

"Why, Huck, you'd have to. How'd you get around it?"

"Why, I just wouldn't stand it. I'd run away."

"Run away! Well, you would be a nice old slouch of a hermit. You'd be a disgrace."

The Red-Handed made no response, being better employed. He had finished gouging out a cob, and now he fitted a weed stem to it, loaded it with tobacco, and was pressing a coal to the charge and blowing a cloud of fragrant smoke -- he was in the full bloom of luxurious contentment. The other pirates envied him this majestic vice, and secretly resolved to acquire it shortly. Presently Huck said:

"What does pirates have to do?"

Tom said:

"Oh, they have just a bully time -- take ships and burn them, and get the money and bury it in awful places in their island where there's ghosts and things to watch it, and kill everybody in the ships -- make 'em walk a plank."

"And they carry the women to the island," said Joe; "they don't kill the women."

"No," assented Tom, "they don't kill the women -- they're too noble. And the women's always beautiful, too."

"And don't they wear the bulliest clothes! Oh no! All gold and silver and di'monds," said Joe, with enthusiasm.

"Who?" said Huck.

"Why, the pirates."

Huck scanned his own clothing forlornly.

"I reckon I ain't dressed fitten for a pirate," said he, with a regretful pathos in his voice; "but I ain't got none but these."

But the other boys told him the fine clothes would come fast enough, after they should have begun their adventures. They made him understand that his poor rags would do to begin with, though it was customary for wealthy pirates to start with a proper wardrobe.

Gradually their talk died out and drowsiness began to steal upon the eyelids of the little waifs. The pipe dropped from the fingers of the Red-Handed, and he slept the sleep of the conscience-free and the weary. The Terror of the Seas and the Black Avenger of the Spanish Main had more difficulty in getting to sleep. They said their prayers inwardly, and lying down, since there was nobody there with authority to make them kneel and recite aloud; in truth, they had a mind not to say them at all, but they were afraid to proceed to such lengths as that, lest they might call down a sudden and special thunderbolt from heaven. Then at once they reached and hovered upon the imminent verge of sleep -- but an intruder came, now, that would not "down." It was conscience. They began to feel a vague fear that they had been doing wrong to run away; and next they thought of the stolen meat, and then the real torture came. They tried to argue it away by reminding conscience that they had purloined sweetmeats and apples scores of times; but conscience was not to be appeased by such thin plausibilities; it seemed to them, in the end, that there was no getting around the stubborn fact that taking sweetmeats was only "hooking," while taking bacon and hams and such valuables was plain simple stealing -- and there was a command against that in the Bible. So they inwardly resolved that so long as they remained in the business, their piracies should not again be sullied with the crime of stealing. Then conscience granted a truce, and these curiously inconsistent pirates fell peacefully to sleep.

The Adventures of Tom Sawyer/Chapter XIV

When Tom awoke in the morning, he wondered where he was. He sat up and rubbed his eyes and looked around. Then he comprehended. It was the cool gray dawn, and there was a delicious sense of repose and peace in the deep pervading calm and silence of the woods. Not a leaf stirred; not a sound obtruded upon great Nature's meditation. Beaded dewdrops stood upon the leaves and grasses. A white layer of ashes covered the fire, and a thin blue breath of smoke rose straight into the air. Joe and Huck still slept.

Now, far away in the woods a bird called; another answered; presently the hammering of a woodpecker was heard. Gradually the cool dim gray of the morning whitened, and as gradually sounds multiplied and life manifested itself. The marvel of Nature shaking off sleep and going to work unfolded itself to the musing boy. A little green worm came crawling over a dewy leaf, lifting two-thirds of his body into the air from time to time and "sniffing around," then proceeding again -- for he was measuring, Tom said; and when the worm approached him, of its own accord, he sat as still as a stone, with his hopes rising and falling, by turns, as the creature still came toward him or seemed inclined to go elsewhere; and when at last it considered a painful moment with its curved body in the air and then came decisively down upon Tom's leg and began a journey over him, his whole heart was glad -- for that meant that he was going to have a new suit of clothes -- without the shadow of a doubt a gaudy piratical uniform. Now a procession of ants appeared, from nowhere in particular, and went about their labors; one struggled manfully by with a dead spider five times as big as itself in its arms, and lugged it straight up a tree-trunk. A brown spotted lady-bug climbed the dizzy height of a grass blade, and Tom bent down close to it and said, "Lady-bug, lady-bug, fly away home, your house is on fire, your children's alone," and she took wing and went off to see about it -- which did not surprise the boy, for he knew of old that this insect was credulous about conflagrations, and he had practised upon its simplicity more than once. A tumblebug came next, heaving sturdily at its ball, and Tom touched the creature, to see it shut its legs against its body and pretend to be dead. The birds were fairly rioting by this time. A catbird, the Northern mocker, lit in a tree over Tom's head, and trilled out her imitations of her neighbors in a rapture of enjoyment; then a shrill jay swept down, a flash of blue flame, and stopped on a twig almost within the boy's reach, cocked his head to one side and eyed the strangers with a consuming curiosity; a gray squirrel and a big fellow of the "fox" kind came scurrying along, sitting up at intervals to inspect and chatter at the boys, for the wild things had probably never seen a human being before and scarcely knew whether to be afraid or not. All Nature was wide awake and stirring, now; long lances of sunlight pierced down through the dense foliage far and near, and a few butterflies came fluttering upon the scene.

Tom stirred up the other pirates and they all clattered away with a shout, and in a minute or two were stripped and chasing after and tumbling over each other in the shallow limpid water of the white sandbar. They felt no longing for the little village sleeping in the distance beyond the majestic waste of water. A vagrant current or a slight rise in the river had carried off their raft, but this only gratified them, since its going was something like burning the bridge between them and civilization.

They came back to camp wonderfully refreshed, glad-hearted, and ravenous; and they soon had the camp-fire blazing up again. Huck found a spring of clear cold water close by, and the boys made cups of broad oak or hickory leaves, and felt that water, sweetened with such a wildwood charm as that, would be a good enough substitute for coffee. While Joe was slicing bacon for breakfast, Tom and Huck asked him to hold on a minute; they stepped to a promising nook in the river-bank and threw in their lines; almost immediately they had reward. Joe had not had time to get impatient before they were back again with some handsome bass, a couple of sun-perch and a small catfish -- provisions enough for quite a family. They fried the fish with the bacon, and were astonished; for no fish had ever seemed so delicious before. They did not know that the quicker a fresh-water fish is on the fire after he is caught the better he is; and they reflected little upon what a sauce open-air sleeping, open-air exercise, bathing, and a large ingredient of hunger make, too.

They lay around in the shade, after breakfast, while Huck had a smoke, and then went off through the woods on an exploring expedition. They tramped gayly along, over decaying logs, through tangled underbrush, among solemn monarchs of the forest, hung from their crowns to the ground with a drooping regalia of grape-vines. Now and then they came upon snug nooks carpeted with grass and jeweled with flowers.

They found plenty of things to be delighted with, but nothing to be astonished at. They discovered that the island was about three miles long and a quarter of a mile wide, and that the shore it lay closest to was only separated from it by a narrow channel hardly two hundred yards wide. They took a swim about every hour, so it was close upon the middle of the afternoon when they got back to camp. They were too hungry to stop to fish, but they fared sumptuously upon cold ham, and then threw themselves down in the shade to talk. But the talk soon began to drag, and then died. The stillness, the solemnity that brooded in the woods, and the sense of loneliness, began to tell upon the spirits of the boys. They fell to thinking. A sort of undefined longing crept upon them. This took dim shape, presently -- it was budding homesickness. Even Finn the Red-Handed was dreaming of his doorsteps and empty hogsheads. But they were all ashamed of their weakness, and none was brave enough to speak his thought.

For some time, now, the boys had been dully conscious of a peculiar sound in the distance, just as one sometimes is of the ticking of a clock which he takes no distinct note of. But now this mysterious sound became more pronounced, and forced a recognition. The boys started, glanced at each other, and then each assumed a listening attitude. There was a long silence, profound and unbroken; then a deep, sullen boom came floating down out of the distance.

"What is it!" exclaimed Joe, under his breath.

"I wonder," said Tom in a whisper.

"Tain't thunder," said Huckleberry, in an awed tone, "becuz thunder --"

"Hark!" said Tom. "Listen -- don't talk."

They waited a time that seemed an age, and then the same muffled boom troubled the solemn hush.

"Let's go and see."

They sprang to their feet and hurried to the shore toward the town. They parted the bushes on the bank and peered out over the water. The little steam ferryboat was about a mile below the village, drifting with the current. Her broad deck seemed crowded with people. There were a great many skiffs rowing about or floating with the stream in the neighborhood of the ferryboat, but the boys could not determine what the men in them were doing. Presently a great jet of white smoke burst from the ferryboat's side, and as it expanded and rose in a lazy cloud, that same dull throb of sound was borne to the listeners again.

"I know now!" exclaimed Tom; "somebody's drowned!"

"That's it!" said Huck; "they done that last summer, when Bill Turner got drowned; they shoot a cannon over the water, and that makes him come up to the top. Yes, and they take loaves of bread and put quicksilver in 'em and set 'em afloat, and wherever there's anybody that's drowned, they'll float right there and stop."

"Yes, I've heard about that," said Joe. "I wonder what makes the bread do that."

"Oh, it ain't the bread, so much," said Tom; "I reckon it's mostly what they say over it before they start it out."

"But they don't say anything over it," said Huck. "I've seen 'em and they don't."

"Well, that's funny," said Tom. "But maybe they say it to themselves. Of course they do. Anybody might know that."

The other boys agreed that there was reason in what Tom said, because an ignorant lump of bread, uninstructed by an incantation, could not be expected to act very intelligently when set upon an errand of such gravity.

"By jings, I wish I was over there, now," said Joe.

"I do too" said Huck "I'd give heaps to know who it is."

The boys still listened and watched. Presently a revealing thought flashed through Tom's mind, and he exclaimed:

"Boys, I know who's drowned -- it's us!"

They felt like heroes in an instant. Here was a gorgeous triumph; they were missed; they were mourned; hearts were breaking on their account; tears were being shed; accusing memories of unkindness to these poor lost lads were rising up, and unavailing regrets and remorse were being indulged; and best of all, the departed were the talk of the whole town, and the envy of all the boys, as far as this dazzling notoriety was concerned. This was fine. It was worth while to be a pirate, after all.

As twilight drew on, the ferryboat went back to her accustomed business and the skiffs disappeared. The pirates returned to camp. They were jubilant with vanity over their new grandeur and the illustrious trouble they were making. They caught fish, cooked supper and ate it, and then fell to guessing at what the village was thinking and saying about them; and the pictures they drew of the public distress on their account were gratifying to look upon -- from their point of view. But when the shadows of night closed them in, they gradually ceased to talk, and sat gazing into the fire, with their minds evidently wandering elsewhere. The excitement was gone, now, and Tom and Joe could not keep back thoughts of certain persons at home who were not enjoying this fine frolic as much as they were. Misgivings came; they grew troubled and unhappy; a sigh or two escaped, unawares. By and by Joe timidly ventured upon a roundabout "feeler" as to how the others might look upon a return to civilization -- not right now, but --

Tom withered him with derision! Huck, being uncommitted as yet, joined in with Tom, and the waverer quickly "explained," and was glad to get out of the scrape with as little taint of chicken-hearted homesickness clinging to his garments as he could. Mutiny was effectually laid to rest for the moment.

As the night deepened, Huck began to nod, and presently to snore. Joe followed next. Tom lay upon his elbow motionless, for some time, watching the two intently. At last he got up cautiously, on his knees, and went searching among the grass and the flickering reflections flung by the camp-fire. He picked up and inspected several large semi-cylinders of the thin white bark of a sycamore, and finally chose two which seemed to suit him. Then he knelt by the fire and painfully wrote something upon each of these with his "red keel"; one he rolled up and put in his jacket pocket, and the other he put in Joe's hat and removed it to a little distance from the owner. And he also put into the hat certain schoolboy treasures of almost inestimable value -- among them a lump of chalk, an India-rubber ball, three fishhooks, and one of that kind of marbles known as a "sure 'nough crystal." Then he tiptoed his way cautiously among the trees till he felt that he was out of hearing, and straightway broke into a keen run in the direction of the sandbar.

The Adventures of Tom Sawyer/Chapter XV

A few minutes later Tom was in the shoal water of the bar, wading toward the Illinois shore. Before the depth reached his middle he was half-way over; the current would permit no more wading, now, so he struck out confidently to swim the remaining hundred yards. He swam quartering upstream, but still was swept downward rather faster than he had expected. However, he reached the shore finally, and drifted along till he found a low place and drew himself out. He put his hand on his jacket pocket, found his piece of bark safe, and then struck through the woods, following the shore, with streaming garments. Shortly before ten o'clock he came out into an open place opposite the village, and saw the ferryboat lying in the shadow of the trees and the high bank. Everything was quiet under the blinking stars. He crept down the bank, watching with all his eyes, slipped into the water, swam three or four strokes and climbed into the skiff that did "yaw!" duty at the boat's stern. He laid himself down under the thwarts and waited, panting.

Presently the cracked bell tapped and a voice gave the order to "cast off." A minute or two later the skiff's head was standing high up, against the boat's swell, and the voyage was begun. Tom felt happy in his success, for he knew it was the boat's last trip for the night. At the end of a long twelve or fifteen minutes the wheels stopped, and Tom slipped overboard and swam ashore in the dusk, landing fifty yards downstream, out of danger of possible stragglers.

He flew along unfrequented alleys, and shortly found himself at his aunt's back fence. He climbed over, approached the "ell," and looked in at the sitting-room window, for a light was burning there. There sat Aunt Polly, Sid, Mary, and Joe Harper's mother, grouped together, talking. They were by the bed, and the bed was between them and the door. Tom went to the door and began to softly lift the latch; then he pressed gently and the door yielded a crack; he continued pushing cautiously, and quaking every time it creaked, till he judged he might squeeze through on his knees; so he put his head through and began, warily.

"What makes the candle blow so?" said Aunt Polly. Tom hurried up. "Why, that door's open, I believe. Why, of course it is. No end of strange things now. Go 'long and shut it, Sid."

Tom disappeared under the bed just in time. He lay and "breathed" himself for a time, and then crept to where he could almost touch his aunt's foot.

"But as I was saying," said Aunt Polly, "he warn't bad, so to say -- only mischeevous. Only just giddy, and harum-scarum, you know. He warn't any more responsible than a colt. He never meant any harm, and he was the best-hearted boy that ever was" -- and she began to cry.

"It was just so with my Joe -- always full of his devilment, and up to every kind of mischief, but he was just as unselfish and kind as he could be -- and laws bless me, to think I went and whipped him for taking that cream, never once recollecting that I throwed it out myself because it was sour, and I never to see him again in this world, never, never, never, poor abused boy!" And Mrs. Harper sobbed as if her heart would break.

"I hope Tom's better off where he is," said Sid, "but if he'd been better in some ways --"

"Sid!" Tom felt the glare of the old lady's eye, though he could not see it. "Not a word against my Tom, now that he's gone! God'll take care of him -- never you trouble yourself, sir! Oh, Mrs. Harper, I don't know how to give him up! I don't know how to give him up! He was such a comfort to me, although he tormented my old heart out of me, 'most."

"The Lord giveth and the Lord hath taken away -- Blessed be the name of the Lord! But it's so hard -- Oh, it's so hard! Only last Saturday my Joe busted a firecracker right under my nose and I knocked him sprawling. Little did I know then, how soon -- Oh, if it was to do over again I'd hug him and bless him for it."

"Yes, yes, yes, I know just how you feel, Mrs. Harper, I know just exactly how you feel. No longer ago than yesterday noon, my Tom took and filled the cat full of Pain-killer, and I did think the cretur would tear the house down. And God forgive me, I cracked Tom's head with my thimble, poor boy, poor dead boy. But he's out of all his troubles now. And the last words I ever heard him say was to reproach --"

But this memory was too much for the old lady, and she broke entirely down. Tom was snuffling, now, himself -- and more in pity of himself than anybody else. He could hear Mary crying, and putting in a kindly word for him from time to time. He began to have a nobler opinion of himself than ever before. Still, he was sufficiently touched by his aunt's grief to long to rush out from under the bed and overwhelm her with joy -- and the theatrical gorgeousness of the thing appealed strongly to his nature, too, but he resisted and lay still.

He went on listening, and gathered by odds and ends that it was conjectured at first that the boys had got drowned while taking a swim; then the small raft had been missed; next, certain boys said the missing lads had promised that the village should "hear something" soon; the wise-heads had "put this and that together" and decided that the lads had gone off on that raft and would turn up at the next town below, presently; but toward noon the raft had been found, lodged against the Missouri shore some five or six miles below the village -- and then hope perished; they must be drowned, else hunger would have driven them home by nightfall if not sooner. It was believed that the search for the bodies had been a fruitless effort merely because the drowning must have occurred in midchannel, since the boys, being good swimmers, would otherwise have escaped to shore. This was Wednesday night. If the bodies continued missing until Sunday, all hope would be given over, and the funerals would be preached on that morning. Tom shuddered.

Mrs. Harper gave a sobbing good-night and turned to go. Then with a mutual impulse the two bereaved women flung themselves into each other's arms and had a good, consoling cry, and then parted. Aunt Polly was tender far beyond her wont, in her good-night to Sid and Mary. Sid snuffled a bit and Mary went off crying with all her heart.

Aunt Polly knelt down and prayed for Tom so touchingly, so appealingly, and with such measureless love in her words and her old trembling voice, that he was weltering in tears again, long before she was through.

He had to keep still long after she went to bed, for she kept making broken-hearted ejaculations from time to time, tossing unrestfully, and turning over. But at last she was still, only moaning a little in her sleep. Now the boy stole out, rose gradually by the bedside, shaded the candle-light with his hand, and stood regarding her. His heart was full of pity for her. He took out his sycamore scroll and placed it by the candle. But something occurred to him, and he lingered considering. His face lighted with a happy solution of his thought; he put the bark hastily in his pocket. Then he bent over and kissed the faded lips, and straightway made his stealthy exit, latching the door behind him.

He threaded his way back to the ferry landing, found nobody at large there, and walked boldly on board the boat, for he knew she was tenantless except that there was a watchman, who always turned in and slept like a graven image. He untied the skiff at the stern, slipped into it, and was soon rowing cautiously upstream. When he had pulled a mile above the village, he started quartering across and bent himself stoutly to his work. He hit the landing on the other side neatly, for this was a familiar bit of work to him. He was moved to capture the skiff, arguing that it might be considered a ship and therefore legitimate prey for a pirate, but he knew a thorough search would be made for it and that might end in revelations. So he stepped ashore and entered the woods.

He sat down and took a long rest, torturing himself meanwhile to keep awake, and then started warily down the home-stretch. The night was far spent. It was broad daylight before he found himself fairly abreast the island bar. He rested again until the sun was well up and gilding the great river with its splendor, and then he plunged into the stream. A little later he paused, dripping, upon the threshold of the camp, and heard Joe say:

"No, Tom's true-blue, Huck, and he'll come back. He won't desert. He knows that would be a disgrace to a pirate, and Tom's too proud for that sort of thing. He's up to something or other. Now I wonder what?"

"Well, the things is ours, anyway, ain't they?"

Pretty near, but not yet, Huck. The writing says they are if he ain't back here to breakfast."

"Which he is!" exclaimed Tom, with fine dramatic effect, stepping grandly into camp.

A sumptuous breakfast of bacon and fish was shortly provided, and as the boys set to work upon it, Tom recounted (and adorned) his adventures. They were a vain and boastful company of heroes when the tale was done. Then Tom hid himself away in a shady nook to sleep till noon, and the other pirates got ready to fish and explore.

The Adventures of Tom Sawyer/Chapter XVI

After dinner all the gang turned out to hunt for turtle eggs on the bar. They went about poking sticks into the sand, and when they found a soft place they went down on their knees and dug with their hands. Sometimes they would take fifty or sixty eggs out of one hole. They were perfectly round white things a trifle smaller than an English walnut. They had a famous fried-egg feast that night, and another on Friday morning.

After breakfast they went whooping and prancing out on the bar, and chased each other round and round, shedding clothes as they went, until they were naked, and then continued the frolic far away up the shoal water of the bar, against the stiff current, which latter tripped their legs from under them from time to time and greatly increased the fun. And now and then they stooped in a group and splashed water in each other's faces with their palms, gradually approaching each other, with averted faces to avoid the strangling sprays, and finally gripping and struggling till the best man ducked his neighbor, and then they all went under in a tangle of white legs and arms and came up blowing, sputtering, laughing, and gasping for breath at one and the same time.

When they were well exhausted, they would run out and sprawl on the dry, hot sand, and lie there and cover themselves up with it, and by and by break for the water again and go through the original performance once more. Finally it occurred to them that their naked skin represented flesh-colored "tights" very fairly; so they drew a ring in the sand and had a circus -- with three clowns in it, for none would yield this proudest post to his neighbor.

Next they got their marbles and played "knucks" and "ring-taw" and "keeps" till that amusement grew stale. Then Joe and Huck had another swim, but Tom would not venture, because he found that in kicking off his trousers he had kicked his string of rattlesnake rattles off his ankle, and he wondered how he had escaped cramp so long without the protection of this mysterious charm. He did not venture again until he had found it, and by that time the other boys were tired and ready to rest. They gradually wandered apart, dropped into the "dumps," and fell to gazing longingly across the wide river to where the village lay drowsing in the sun. Tom found himself writing "Becky" in the sand with his big toe; he scratched it out, and was angry with himself for his weakness. But he wrote it again, nevertheless; he could not help it. He erased it once more and then took himself out of temptation by driving the other boys together and joining them.

But Joe's spirits had gone down almost beyond resurrection. He was so homesick that he could hardly endure the misery of it. The tears lay very near the surface. Huck was melancholy, too. Tom was downhearted, but tried hard not to show it. He had a secret which he was not ready to tell, yet, but if this mutinous depression was not broken up soon, he would have to bring it out. He said, with a great show of cheerfulness:

"I bet there's been pirates on this island before, boys. We'll explore it again. They've hid treasures here somewhere. How'd you feel to light on a rotten chest full of gold and silver -- hey?"

But it roused only faint enthusiasm, which faded out, with no reply. Tom tried one or two other seductions; but they failed, too. It was discouraging work. Joe sat poking up the sand with a stick and looking very gloomy. Finally he said:

"Oh, boys, let's give it up. I want to go home. It's so lonesome."

"Oh no, Joe, you'll feel better by and by," said Tom. "Just think of the fishing that's here."

"I don't care for fishing. I want to go home."

"But, Joe, there ain't such another swimming-place anywhere."

"Swimming's no good. I don't seem to care for it, somehow, when there ain't anybody to say I sha'n't go in. I mean to go home."

"Oh, shucks! Baby! You want to see your mother, I reckon."

"Yes, I do want to see my mother -- and you would, too, if you had one. I ain't any more baby than you are." And Joe snuffled a little.

"Well, we'll let the cry-baby go home to his mother, won't we, Huck? Poor thing -- does it want to see its mother? And so it shall. You like it here, don't you, Huck? We'll stay, won't we?"

Huck said, "Y-e-s" -- without any heart in it.

"I'll never speak to you again as long as I live," said Joe, rising. "There now!" And he moved moodily away and began to dress himself.

"Who cares!" said Tom. "Nobody wants you to. Go 'long home and get laughed at. Oh, you're a nice pirate. Huck and me ain't cry-babies. We'll stay, won't we, Huck? Let him go if he wants to. I reckon we can get along without him, per'aps."

But Tom was uneasy, nevertheless, and was alarmed to see Joe go sullenly on with his dressing. And then it was discomforting to see Huck eying Joe's preparations so wistfully, and keeping up such an ominous silence. Presently, without a parting word, Joe began to wade off toward the Illinois shore. Tom's heart began to sink. He glanced at Huck. Huck could not bear the look, and dropped his eyes. Then he said:

"I want to go, too, Tom. It was getting so lonesome anyway, and now it'll be worse. Let's us go, too, Tom."

"I won't! You can all go, if you want to. I mean to stay."

"Tom, I better go."

"Well, go 'long -- who's hendering you."

Huck began to pick up his scattered clothes. He said:

"Tom, I wisht you'd come, too. Now you think it over. We'll wait for you when we get to shore."

"Well, you'll wait a blame long time, that's all."

Huck started sorrowfully away, and Tom stood looking after him, with a strong desire tugging at his heart to yield his pride and go along too. He hoped the boys would stop, but they still waded slowly on. It suddenly dawned on Tom that it was become very lonely and still. He made one final struggle with his pride, and then darted after his comrades, yelling:

"Wait! Wait! I want to tell you something!"

They presently stopped and turned around. When he got to where they were, he began unfolding his secret, and they listened moodily till at last they saw the "point" he was driving at, and then they set up a war-whoop of applause and said it was "splendid!" and said if he had told them at first, they wouldn't have started away. He made a plausible excuse; but his real reason had been the fear that not even the secret would keep them with him any very great length of time, and so he had meant to hold it in reserve as a last seduction.

The lads came gayly back and went at their sports again with a will, chattering all the time about Tom's stupendous plan and admiring the genius of it. After a dainty egg and fish dinner, Tom said he wanted to learn to smoke, now. Joe caught at the idea and said he would like to try, too. So Huck made pipes and filled them. These novices had never smoked anything before but cigars made of grape-vine, and they "bit" the tongue, and were not considered manly anyway.

Now they stretched themselves out on their elbows and began to puff, charily, and with slender confidence. The smoke had an unpleasant taste, and they gagged a little, but Tom said:

"Why, it's just as easy! If I'd a knowed this was all, I'd a learnt long ago."

"So would I," said Joe. "It's just nothing."

"Why, many a time I've looked at people smoking, and thought well I wish I could do that; but I never thought I could," said Tom.

"That's just the way with me, hain't it, Huck? You've heard me talk just that way -- haven't you, Huck? I'll leave it to Huck if I haven't."

"Yes -- heaps of times," said Huck.

"Well, I have too," said Tom; "oh, hundreds of times. Once down by the slaughter-house. Don't you remember, Huck? Bob Tanner was there, and Johnny Miller, and Jeff Thatcher, when I said it. Don't you remember, Huck, 'bout me saying that?"

"Yes, that's so," said Huck. "That was the day after I lost a white alley. No, 'twas the day before."

"There -- I told you so," said Tom. "Huck recollects it."

"I bleeve I could smoke this pipe all day," said Joe. "I don't feel sick."

"Neither do I," said Tom. "I could smoke it all day. But I bet you Jeff Thatcher couldn't."

"Jeff Thatcher! Why, he'd keel over just with two draws. Just let him try it once. He'd see!"

"I bet he would. And Johnny Miller -- I wish could see Johnny Miller tackle it once."

"Oh, don't I!" said Joe. "Why, I bet you Johnny Miller couldn't any more do this than nothing. Just one little snifter would fetch him."

"Deed it would, Joe. Say -- I wish the boys could see us now."

"So do I."

"Say -- boys, don't say anything about it, and some time when they're around, I'll come up to you and say, 'Joe, got a pipe? I want a smoke.' And you'll say, kind of careless like, as if it warn't anything, you'll say, 'Yes, I got my old pipe, and another one, but my tobacker ain't very good.' And I'll say, 'Oh, that's all right, if it's strong enough.' And then you'll out with the pipes, and we'll light up just as ca'm, and then just see 'em look!"

"By jings, that'll be gay, Tom! I wish it was Now!"

"So do I! And when we tell 'em we learned when we was off pirating, won't they wish they'd been along?"

"Oh, I reckon not! I'll just bet they will!"

So the talk ran on. But presently it began to flag a trifle, and grow disjointed. The silences widened; the expectoration marvellously increased. Every pore inside the boys' cheeks became a spouting fountain; they could scarcely bail out the cellars under their tongues fast enough to prevent an inundation; little overflowings down their throats occurred in spite of all they could do, and sudden retchings followed every time. Both boys were looking very pale and miserable, now. Joe's pipe dropped from his nerveless fingers. Tom's followed. Both fountains were going furiously and both pumps bailing with might and main. Joe said feebly:

"I've lost my knife. I reckon I better go and find it."

Tom said, with quivering lips and halting utterance:

"I'll help you. You go over that way and I'll hunt around by the spring. No, you needn't come, Huck -- we can find it."

So Huck sat down again, and waited an hour. Then he found it lonesome, and went to find his comrades. They were wide apart in the woods, both very pale, both fast asleep. But something informed him that if they had had any trouble they had got rid of it.

They were not talkative at supper that night. They had a humble look, and when Huck prepared his pipe after the meal and was going to prepare theirs, they said no, they were not feeling very well -- something they ate at dinner had disagreed with them.

About midnight Joe awoke, and called the boys. There was a brooding oppressiveness in the air that seemed to bode something. The boys huddled themselves together and sought the friendly companionship of the fire, though the dull dead heat of the breathless atmosphere was stifling. They sat still, intent and waiting. The solemn hush continued. Beyond the light of the fire everything was swallowed up in the blackness of darkness. Presently there came a quivering glow that vaguely revealed the foliage for a moment and then vanished. By and by another came, a little stronger. Then another. Then a faint moan came sighing through the branches of the forest and the boys felt a fleeting breath upon their cheeks, and shuddered with the fancy that the Spirit of the Night had gone by. There was a

pause. Now a weird flash turned night into day and showed every little grass-blade, separate and distinct, that grew about their feet. And it showed three white, startled faces, too. A deep peal of thunder went rolling and tumbling down the heavens and lost itself in sullen rumblings in the distance. A sweep of chilly air passed by, rustling all the leaves and snowing the flaky ashes broadcast about the fire. Another fierce glare lit up the forest and an instant crash followed that seemed to rend the tree-tops right over the boys' heads. They clung together in terror, in the thick gloom that followed. A few big rain-drops fell pattering upon the leaves.

"Quick! boys, go for the tent!" exclaimed Tom.

They sprang away, stumbling over roots and among vines in the dark, no two plunging in the same direction. A furious blast roared through the trees, making everything sing as it went. One blinding flash after another came, and peal on peal of deafening thunder. And now a drenching rain poured down and the rising hurricane drove it in sheets along the ground. The boys cried out to each other, but the roaring wind and the booming thunder-blasts drowned their voices utterly. However, one by one they straggled in at last and took shelter under the tent, cold, scared, and streaming with water; but to have company in misery seemed something to be grateful for. They could not talk, the old sail flapped so furiously, even if the other noises would have allowed them. The tempest rose higher and higher, and presently the sail tore loose from its fastenings and went winging away on the blast. The boys seized each others' hands and fled, with many tumblings and bruises, to the shelter of a great oak that stood upon the river-bank. Now the battle was at its highest. Under the ceaseless conflagration of lightning that flamed in the skies, everything below stood out in clean-cut and shadowless distinctness: the bending trees, the billowy river, white with foam, the driving spray of spume-flakes, the dim outlines of the high bluffs on the other side, glimpsed through the drifting cloud-rack and the slanting veil of rain. Every little while some giant tree yielded the fight and fell crashing through the younger growth; and the unflagging thunder-peals came now in ear-splitting explosive bursts, keen and sharp, and unspeakably appalling. The storm culminated in one matchless effort that seemed likely to tear the island to pieces, burn it up, drown it to the tree-tops, blow it away, and deafen every creature in it, all at one and the same moment. It was a wild night for homeless young heads to be out in.

But at last the battle was done, and the forces retired with weaker and weaker threatenings and grumblings, and peace resumed her sway. The boys went back to camp, a good deal awed; but they found there was still something to be thankful for, because the great sycamore, the shelter of their beds, was a ruin, now, blasted by the lightnings, and they were not under it when the catastrophe happened.

Everything in camp was drenched, the camp-fire as well; for they were but heedless lads, like their generation, and had made no provision against rain. Here was matter for dismay, for they were soaked through and chilled. They were eloquent in their distress; but they presently discovered that the fire had eaten so far up under the great log it had been built against (where it curved upward and separated itself from the ground), that a handbreadth or so of it had escaped wetting; so they patiently wrought until, with shreds and bark gathered from the under sides of sheltered logs, they coaxed the fire to burn again. Then they piled on great dead boughs till they had a roaring furnace, and were glad-hearted once more. They dried their boiled ham and had a feast, and after that they sat by the fire and expanded and glorified their midnight adventure until morning, for there was not a dry spot to sleep on, anywhere around.

As the sun began to steal in upon the boys, drowsiness came over them, and they went out on the sandbar and lay down to sleep. They got scorched out by and by, and drearily set about getting breakfast. After the meal they felt rusty, and stiff-jointed, and a little homesick once more. Tom saw the signs, and fell to cheering up the pirates as well as he could. But they cared nothing for marbles, or circus, or swimming, or anything. He reminded them of the imposing secret, and raised a ray of cheer. While it lasted, he got them interested in a new device. This was to knock off being pirates, for a while, and be Indians for a change. They were attracted by this idea; so it was not long before they were stripped, and striped from head to heel with black mud, like so many zebras -- all of them chiefs, of course -- and then they went tearing through the woods to attack an English settlement.

By and by they separated into three hostile tribes, and darted upon each other from ambush with dreadful war-whoops, and killed and scalped each other by thousands. It was a gory day. Consequently it was an extremely satisfactory one.

They assembled in camp toward supper-time, hungry and happy; but now a difficulty arose -- hostile Indians could not break the bread of hospitality together without first making peace, and this was a simple impossibility without smoking a pipe of peace. There was no other process that ever they had heard of. Two of the savages almost wished they had remained pirates. However, there was no other way; so with such show of cheerfulness as they could muster they called for the pipe and took their whiff as it passed, in due form.

And behold, they were glad they had gone into savagery, for they had gained something; they found that they could now smoke a little without having to go and hunt for a lost knife; they did not get sick enough to be seriously uncomfortable. They were not likely to fool away this high promise for lack of effort. No, they practised cautiously, after supper, with right fair success, and so they spent a jubilant evening. They were prouder and happier in their new acquirement than they would have been in the scalping and skinning of the Six Nations. We will leave them to smoke and chatter and brag, since we have no further use for them at present.

The Adventures of Tom Sawyer/Chapter XVII

But there was no hilarity in the little town that same tranquil Saturday afternoon. The Harpers, and Aunt Polly's family, were being put into mourning, with great grief and many tears. An unusual quiet possessed the village, although it was ordinarily quiet enough, in all conscience. The villagers conducted their concerns with an absent air, and talked little; but they sighed often. The Saturday holiday seemed a burden to the children. They had no heart in their sports, and gradually gave them up.

In the afternoon Becky Thatcher found herself moping about the deserted schoolhouse yard, and feeling very melancholy. But she found nothing there to comfort her. She soliloquized:

"Oh, if I only had a brass and iron-knob again! But I haven't got anything now to remember him by." And she choked back a little sob.

Presently she stopped, and said to herself:

"It was right here. Oh, if it was to do over again, I wouldn't say that -- I wouldn't say it for the whole world. But he's gone now; I'll never, never, never see him any more."

This thought broke her down, and she wandered away, with tears rolling down her cheeks. Then quite a group of boys and girls -- playmates of Tom's and Joe's -- came by, and stood looking over the paling fence and talking in reverent tones of how Tom did so-and-so the last time they saw him, and how Joe said this and that small trifle (pregnant with awful prophecy, as they could easily see now!) -- and each speaker pointed out the exact spot where the lost lads stood at the time, and then added something like "and I was a-standing just so -- just as I am now, and as if you was him -- I was as close as that -- and he smiled, just this way -- and then something seemed to go all over me, like -- awful, you know -- and I never thought what it meant, of course, but I can see now!"

Then there was a dispute about who saw the dead boys last in life, and many claimed that dismal distinction, and offered evidences, more or less tampered with by the witness; and when it was ultimately decided who did see the departed last, and exchanged the last words with them, the lucky parties took upon themselves a sort of sacred importance, and were gaped at and envied by all the rest. One poor chap, who had no other grandeur to offer, said with tolerably manifest pride in the remembrance:

"Well, Tom Sawyer he licked me once."

But that bid for glory was a failure. Most of the boys could say that, and so that cheapened the distinction too much. The group loitered away, still recalling memories of the lost heroes, in awed voices.

When the Sunday-school hour was finished, the next morning, the bell began to toll, instead of ringing in the usual way. It was a very still Sabbath, and the mournful sound seemed in keeping with the musing hush that lay upon nature. The villagers began to gather, loitering a moment in the vestibule to converse in whispers about the sad event. But there was no whispering in the house; only the funereal rustling of dresses as the women gathered to their seats disturbed the silence there. None could remember when the little church had been so full before. There was finally a waiting pause, an expectant dumbness, and then Aunt Polly entered, followed by Sid and Mary, and they by the Harper family, all in deep black, and the whole congregation, the old minister as well, rose reverently and stood until the mourners were seated in the front pew. There was another communing silence, broken at intervals by muffled sobs, and then the minister spread his hands abroad and prayed. A moving hymn was sung, and the text followed: "I am the Resurrection and the Life."

As the service proceeded, the clergyman drew such pictures of the graces, the winning ways, and the rare promise of the lost lads that every soul there, thinking he recognized these pictures, felt a pang in remembering that he had persistently blinded himself to them always before, and had as persistently seen only faults and flaws in the poor boys. The minister related many a touching incident in the lives of the departed, too, which illustrated their sweet, generous natures, and the people could easily see, now, how noble and beautiful those episodes were, and remembered with grief that at the time they occurred they had seemed rank rascalities, well deserving of the cowhide. The congregation became more and more moved, as the pathetic tale went on, till at last the whole company broke down and joined the weeping mourners in a chorus of anguished sobs, the preacher himself giving way to his feelings, and crying in the pulpit.

There was a rustle in the gallery, which nobody noticed; a moment later the church door creaked; the minister raised his streaming eyes above his handkerchief, and stood transfixed! First one and then another pair of eyes followed the minister's, and then almost with one impulse the congregation rose and stared while the three dead boys came marching up the aisle, Tom in the lead, Joe next, and Huck, a ruin of drooping rags, sneaking sheepishly in the rear! They had been hid in the unused gallery listening to their own funeral sermon!

Aunt Polly, Mary, and the Harpers threw themselves upon their restored ones, smothered them with kisses and poured out thanksgivings, while poor Huck stood abashed and uncomfortable, not knowing exactly what to do or where to hide from so many unwelcoming eyes. He wavered, and started to slink away, but Tom seized him and said:

"Aunt Polly, it ain't fair. Somebody's got to be glad to see Huck."

"And so they shall. I'm glad to see him, poor motherless thing!" And the loving attentions Aunt Polly lavished upon him were the one thing capable of making him more uncomfortable than he was before.

Suddenly the minister shouted at the top of his voice: "Praise God from whom all blessings flow -- Sing! -- and put your hearts in it!"

And they did. Old Hundred swelled up with a triumphant burst, and while it shook the rafters Tom Sawyer the Pirate looked around upon the envying juveniles about him and confessed in his heart that this was the proudest moment of his life.

As the "sold" congregation trooped out they said they would almost be willing to be made ridiculous again to hear Old Hundred sung like that once more.

Tom got more cuffs and kisses that day -- according to Aunt Polly's varying moods -- than he had earned before in a year; and he hardly knew which expressed the most gratefulness to God and affection for himself.

The Adventures of Tom Sawyer/Chapter XVIII

That was Tom's great secret -- the scheme to return home with his brother pirates and attend their own funerals. They had paddled over to the Missouri shore on a log, at dusk on Saturday, landing five or six miles below the village; they had slept in the woods at the edge of the town till nearly daylight, and had then crept through back lanes and alleys and finished their sleep in the gallery of the church among a chaos of invalided benches.

At breakfast, Monday morning, Aunt Polly and Mary were very loving to Tom, and very attentive to his wants. There was an unusual amount of talk. In the course of it Aunt Polly said:

"Well, I don't say it wasn't a fine joke, Tom, to keep everybody suffering 'most a week so you boys had a good time, but it is a pity you could be so hard-hearted as to let me suffer so. If you could come over on a log to go to your funeral, you could have come over and give me a hint some way that you warn't dead, but only run off."

"Yes, you could have done that, Tom," said Mary; "and I believe you would if you had thought of it."

"Would you, Tom?" said Aunt Polly, her face lighting wistfully. "Say, now, would you, if you'd thought of it?"

"I -- well, I don't know. 'Twould 'a' spoiled everything."

"Tom, I hoped you loved me that much," said Aunt Polly, with a grieved tone that discomforted the boy. "It would have been something if you'd cared enough to think of it, even if you didn't do it."

"Now, auntie, that ain't any harm," pleaded Mary; "it's only Tom's giddy way -- he is always in such a rush that he never thinks of anything."

"More's the pity. Sid would have thought. And Sid would have come and done it, too. Tom, you'll look back, some day, when it's too late, and wish you'd cared a little more for me when it would have cost you so little."

"Now, auntie, you know I do care for you," said Tom.

"I'd know it better if you acted more like it."

"I wish now I'd thought," said Tom, with a repentant tone; "but I dreamt about you, anyway. That's something, ain't it?"

"It ain't much -- a cat does that much -- but it's better than nothing. What did you dream?"

"Why, Wednesday night I dreamt that you was sitting over there by the bed, and Sid was sitting by the woodbox, and Mary next to him."

"Well, so we did. So we always do. I'm glad your dreams could take even that much trouble about us."

"And I dreamt that Joe Harper's mother was here."

"Why, she was here! Did you dream any more?"

"Oh, lots. But it's so dim, now."

"Well, try to recollect -- can't you?"

"Somehow it seems to me that the wind -- the wind blowed the -- the --"

"Try harder, Tom! The wind did blow something. Come!"

Tom pressed his fingers on his forehead an anxious minute, and then said:

"I've got it now! I've got it now! It blowed the candle!"

"Mercy on us! Go on, Tom -- go on!"

"And it seems to me that you said, 'Why, I believe that that door --'"

"Go on, Tom!"

"Just let me study a moment -- just a moment. Oh, yes -- you said you believed the door was open."

"As I'm sitting here, I did! Didn't I, Mary! Go on!"

"And then -- and then -- well I won't be certain, but it seems like as if you made Sid go and -- and --"

"Well? Well? What did I make him do, Tom? What did I make him do?"

"You made him -- you -- Oh, you made him shut it."

"Well, for the land's sake! I never heard the beat of that in all my days! Don't tell me there ain't anything in dreams, any more. Sereny Harper shall know of this before I'm an hour older. I'd like to see her get around this with her rubbage 'bout superstition. Go on, Tom!"

"Oh, it's all getting just as bright as day, now. Next you said I warn't bad, only mischeevous and harum-scarum, and not any more responsible than -- than -- I think it was a colt, or something."

"And so it was! Well, goodness gracious! Go on, Tom!"

"And then you began to cry."

"So I did. So I did. Not the first time, neither. And then --"

"Then Mrs. Harper she began to cry, and said Joe was just the same, and she wished she hadn't whipped him for taking cream when she'd throwed it out her own self --"

"Tom! The sperrit was upon you! You was a prophesying -- that's what you was doing! Land alive, go on, Tom!"

"Then Sid he said -- he said --"

"I don't think I said anything," said Sid.

"Yes you did, Sid," said Mary.

"Shut your heads and let Tom go on! What did he say, Tom?"

"He said -- I think he said he hoped I was better off where I was gone to, but if I'd been better sometimes --"

"There, d'you hear that! It was his very words!"

"And you shut him up sharp."

"I lay I did! There must 'a' been an angel there. There was an angel there, somewheres!"

"And Mrs. Harper told about Joe scaring her with a firecracker, and you told about Peter and the Painkiller --"

"Just as true as I live!"

"And then there was a whole lot of talk 'bout dragging the river for us, and 'bout having the funeral Sunday, and then you and old Miss Harper hugged and cried, and she went."

"It happened just so! It happened just so, as sure as I'm a-sitting in these very tracks. Tom, you couldn't told it more like if you'd 'a' seen it! And then what? Go on, Tom!"

"Then I thought you prayed for me -- and I could see you and hear every word you said. And you went to bed, and I was so sorry that I took and wrote on a piece of sycamore bark, 'We ain't dead -- we are only off being pirates,' and put it on the table by the candle; and then you looked so good, laying there asleep, that I thought I went and leaned over and kissed you on the lips."

"Did you, Tom, did you! I just forgive you everything for that!" And she seized the boy in a crushing embrace that made him feel like the guiltiest of villains.

"It was very kind, even though it was only a -- dream," Sid soliloquized just audibly.

"Shut up, Sid! A body does just the same in a dream as he'd do if he was awake. Here's a big Milum apple I've been saving for you, Tom, if you was ever found again -- now go 'long to school. I'm thankful to the good God and Father of us all I've got you back, that's long-suffering and merciful to them that believe on Him and keep His word, though goodness knows I'm unworthy of it, but if only the worthy ones got His blessings and had His hand to help them over the rough places, there's few enough would smile here or ever enter into His rest when the long night comes. Go 'long Sid, Mary, Tom -- take yourselves off -- you've hendered me long enough."

The children left for school, and the old lady to call on Mrs. Harper and vanquish her realism with Tom's marvellous dream. Sid had better judgment than to utter the thought that was in his mind as he left the house. It was this: "Pretty thin -- as long a dream as that, without any mistakes in it!"

What a hero Tom was become, now! He did not go skipping and prancing, but moved with a dignified swagger as became a pirate who felt that the public eye was on him. And indeed it was; he tried not to seem to see the looks or hear the remarks as he passed along, but they were food and drink to him. Smaller boys than himself flocked at his heels, as proud to be seen with him, and tolerated by him, as if he had been the drummer at the head of a procession or the elephant leading a menagerie into town. Boys of his own size pretended not to know he had been away at all; but they were consuming with envy, nevertheless. They would have given anything to have that swarthy suntanned skin of his, and his glittering notoriety; and Tom would not have parted with either for a circus.

At school the children made so much of him and of Joe, and delivered such eloquent admiration from their eyes, that the two heroes were not long in becoming insufferably "stuck-up." They began to tell their adventures to hungry listeners -- but they only began; it was not a thing likely to have an end, with imaginations like theirs to furnish material. And finally, when they got out their pipes and went serenely puffing around, the very summit of glory was reached.

Tom decided that he could be independent of Becky Thatcher now. Glory was sufficient. He would live for glory. Now that he was distinguished, maybe she would be wanting to "make up." Well, let her -- she should see that he could be as indifferent as some other people. Presently she arrived. Tom pretended not to see her. He moved away and joined a group of boys and girls and began to talk. Soon he observed that she was tripping gayly back and forth with flushed face and dancing eyes, pretending to be busy chasing schoolmates, and screaming with laughter when she made a capture; but he noticed that she always made her captures in his vicinity, and that she seemed to cast a conscious eye in his direction at such times, too. It gratified all the vicious vanity that was in him; and so, instead of winning him, it only "set him up" the more and made him the more diligent to avoid betraying that he knew she was about. Presently she gave over skylarking, and moved irresolutely about, sighing once or twice and glancing furtively and wistfully toward Tom. Then she observed that now Tom was talking more particularly to Amy Lawrence than to any one else. She felt a sharp pang and grew disturbed and uneasy at once. She tried to go away, but her feet were treacherous, and carried her to the group instead. She said to a girl almost at Tom's elbow -- with sham vivacity:

"Why, Mary Austin! you bad girl, why didn't you come to Sunday-school?"

"I did come -- didn't you see me?"

"Why, no! Did you? Where did you sit?"

"I was in Miss Peters' class, where I always go. I saw you."

"Did you? Why, it's funny I didn't see you. I wanted to tell you about the picnic."

"Oh, that's jolly. Who's going to give it?"

"My ma's going to let me have one."

"Oh, goody; I hope she'll let me come."

"Well, she will. The picnic's for me. She'll let anybody come that I want, and I want you."

"That's ever so nice. When is it going to be?"

"By and by. Maybe about vacation."

"Oh, won't it be fun! You going to have all the girls and boys?"

"Yes, every one that's friends to me -- or wants to be"; and she glanced ever so furtively at Tom, but he talked right along to Amy Lawrence about the terrible storm on the island, and how the lightning tore the great sycamore tree "all to flinders" while he was "standing within three feet of it."

"Oh, may I come?" said Grace Miller.

"Yes."

"And me?" said Sally Rogers.

"Yes."

"And me, too?" said Susy Harper. "And Joe?"

"Yes."

And so on, with clapping of joyful hands till all the group had begged for invitations but Tom and Amy. Then Tom turned coolly away, still talking, and took Amy with him. Becky's lips trembled and the tears came to her eyes; she hid these signs with a forced gayety and went on chattering, but the life had gone out of the picnic, now, and out of everything else; she got away as soon as she could and hid herself and had what her sex call "a good cry." Then she sat moody, with wounded pride, till the bell rang. She roused up, now, with a vindictive cast in her eye, and gave her plaited tails a shake and said she knew what she'd do.

At recess Tom continued his flirtation with Amy with jubilant self-satisfaction. And he kept drifting about to find Becky and lacerate her with the performance. At last he spied her, but there was a sudden falling of his mercury. She was sitting cosily on a little bench behind the schoolhouse looking at a picture-book with Alfred Temple -- and so absorbed were they, and their heads so close together over the book, that they did not seem to be conscious of anything in the world besides. Jealousy ran red-hot through Tom's veins. He began to hate himself for throwing away the chance Becky had offered for a reconciliation. He called himself a fool, and all the hard names he could think of. He wanted to cry with vexation. Amy chatted happily along, as they walked, for her heart was singing, but Tom's tongue had lost its function. He did not hear what Amy was saying, and whenever she paused expectantly he could only stammer an awkward assent, which was as often misplaced as otherwise. He kept drifting to the rear of the schoolhouse, again and again, to sear his eyeballs with the hateful spectacle there. He could not help it. And it maddened him to see, as he thought he saw, that Becky Thatcher never once suspected that he was even in the land of the living. But she did see, nevertheless; and she knew she was winning her fight, too, and was glad to see him suffer as she had suffered.

Amy's happy prattle became intolerable. Tom hinted at things he had to attend to; things that must be done; and time was fleeting. But in vain -- the girl chirped on. Tom thought, "Oh, hang her, ain't I ever going to get rid of her?" At last he must be attending to those things -- and she said artlessly that she would be "around" when school let out. And he hastened away, hating her for it.

"Any other boy!" Tom thought, grating his teeth. "Any boy in the whole town but that Saint Louis smarty that thinks he dresses so fine and is aristocracy! Oh, all right, I licked you the first day you ever saw this town, mister, and I'll lick you again! You just wait till I catch you out! I'll just take and --"

And he went through the motions of thrashing an imaginary boy -- pummelling the air, and kicking and gouging. "Oh, you do, do you? You holler 'nough, do you? Now, then, let that learn you!" And so the imaginary flogging was finished to his satisfaction.

Tom fled home at noon. His conscience could not endure any more of Amy's grateful happiness, and his jealousy could bear no more of the other distress. Becky resumed her picture inspections with Alfred, but as the minutes dragged along and no Tom came to suffer, her triumph began to cloud and she lost interest; gravity and absent-mindedness followed, and then melancholy; two or three times she pricked up her ear at a footstep, but it was a false hope; no Tom came. At last she grew entirely miserable and wished she hadn't carried it so far. When poor Alfred, seeing that he was losing her, he did not know how, kept exclaiming: "Oh, here's a jolly one! look at this!" she lost patience at last, and said, "Oh, don't bother me! I don't care for them!" and burst into tears, and got up and walked away.

Alfred dropped alongside and was going to try to comfort her, but she said:

"Go away and leave me alone, can't you! I hate you!"

So the boy halted, wondering what he could have done -- for she had said she would look at pictures all through the nooning -- and she walked on, crying. Then Alfred went musing into the deserted schoolhouse. He was humiliated and angry. He easily guessed his way to the truth -- the girl had simply made a convenience of him to vent her spite upon Tom Sawyer. He was far from hating Tom the less when this thought occurred to him. He wished there was some way to get that boy into trouble without much risk to himself. Tom's spelling-book fell under his eye. Here was his opportunity. He gratefully opened to the lesson for the afternoon and poured ink upon the page.

Becky, glancing in at a window behind him at the moment, saw the act, and moved on, without discovering herself. She started homeward, now, intending to find Tom and tell him; Tom would be thankful and their troubles would be healed. Before she was half way home, however, she had changed her mind. The thought of Tom's treatment of her when she was talking about her picnic came scorching back and filled her with shame. She resolved to let him get whipped on the damaged spelling-book's account, and to hate him forever, into the bargain.

The Adventures of Tom Sawyer/Chapter XIX

Tom arrived at home in a dreary mood, and the first thing his aunt said to him showed him that he had brought his sorrows to an unpromising market:

"Tom, I've a notion to skin you alive!"

"Auntie, what have I done?"

"Well, you've done enough. Here I go over to Sereny Harper, like an old softy, expecting I'm going to make her believe all that rubbage about that dream, when lo and behold you she'd found out from Joe that you was over here and heard all the talk we had that night. Tom, I don't know what is to become of a boy that will act like that. It makes me feel so bad to think you could let me go to Sereny Harper and make such a fool of myself and never say a word."

This was a new aspect of the thing. His smartness of the morning had seemed to Tom a good joke before, and very ingenious. It merely looked mean and shabby now. He hung his head and could not think of anything to say for a moment. Then he said:

"Auntie, I wish I hadn't done it -- but I didn't think."

"Oh, child, you never think. You never think of anything but your own selfishness. You could think to come all the way over here from Jackson's Island in the night to laugh at our troubles, and you could think to fool me with a lie about a dream; but you couldn't ever think to pity us and save us from sorrow."

"Auntie, I know now it was mean, but I didn't mean to be mean. I didn't, honest. And besides, I didn't come over here to laugh at you that night."

"What did you come for, then?"

"It was to tell you not to be uneasy about us, because we hadn't got drowned."

"Tom, Tom, I would be the thankfulest soul in this world if I could believe you ever had as good a thought as that, but you know you never did -- and I know it, Tom."

"Indeed and 'deed I did, auntie -- I wish I may never stir if I didn't."

"Oh, Tom, don't lie -- don't do it. It only makes things a hundred times worse."

"It ain't a lie, auntie; it's the truth. I wanted to keep you from grieving -- that was all that made me come."

"I'd give the whole world to believe that -- it would cover up a power of sins, Tom. I'd 'most be glad you'd run off and acted so bad. But it ain't reasonable; because, why didn't you tell me, child?"

"Why, you see, when you got to talking about the funeral, I just got all full of the idea of our coming and hiding in the church, and I couldn't somehow bear to spoil it. So I just put the bark back in my pocket and kept mum."

"What bark?"

"The bark I had wrote on to tell you we'd gone pirating. I wish, now, you'd waked up when I kissed you -- I do, honest."

The hard lines in his aunt's face relaxed and a sudden tenderness dawned in her eyes.

"Did you kiss me, Tom?"

"Why, yes, I did."

"Are you sure you did, Tom?"

"Why, yes, I did, auntie -- certain sure."

"What did you kiss me for, Tom?"

"Because I loved you so, and you laid there moaning and I was so sorry."

The words sounded like truth. The old lady could not hide a tremor in her voice when she said:

"Kiss me again, Tom! -- and be off with you to school, now, and don't bother me any more."

The moment he was gone, she ran to a closet and got out the ruin of a jacket which Tom had gone pirating in. Then she stopped, with it in her hand, and said to herself:

"No, I don't dare. Poor boy, I reckon he's lied about it -- but it's a blessed, blessed lie, there's such a comfort come from it. I hope the Lord -- I know the Lord will forgive him, because it was such good-heartedness in him to tell it. But I don't want to find out it's a lie. I won't look."

She put the jacket away, and stood by musing a minute. Twice she put out her hand to take the garment again, and twice she refrained. Once more she ventured, and this time she fortified herself with the thought: "It's a good lie -- it's a good lie -- I won't let it grieve me." So she sought the jacket pocket. A moment later she was reading Tom's piece of bark through flowing tears and saying: "I could forgive the boy, now, if he'd committed a million sins!"

The Adventures of Tom Sawyer/Chapter XX

There was something about Aunt Polly's manner, when she kissed Tom, that swept away his low spirits and made him light-hearted and happy again. He started to school and had the luck of coming upon Becky Thatcher at the head of Meadow Lane. His mood always determined his manner. Without a moment's hesitation he ran to her and said:

"I acted mighty mean to-day, Becky, and I'm so sorry. I won't ever, ever do that way again, as long as ever I live -- please make up, won't you?"

The girl stopped and looked him scornfully in the face:

"I'll thank you to keep yourself to yourself, Mr. Thomas Sawyer. I'll never speak to you again."

She tossed her head and passed on. Tom was so stunned that he had not even presence of mind enough to say "Who cares, Miss Smarty?" until the right time to say it had gone by. So he said nothing. But he was in a fine rage, nevertheless. He moped into the schoolyard wishing she were a boy, and imagining how he would trounce her if she were. He presently encountered her and delivered a stinging remark as he passed. She hurled one in return, and the angry breach was complete. It seemed to Becky, in her hot resentment, that she could hardly wait for school to "take in," she was so impatient to see Tom flogged for the injured spelling-book. If she had had any lingering notion of exposing Alfred Temple, Tom's offensive fling had driven it entirely away.

Poor girl, she did not know how fast she was nearing trouble herself. The master, Mr. Dobbins, had reached middle age with an unsatisfied ambition. The darling of his desires was, to be a doctor, but poverty had decreed that he should be nothing higher than a village schoolmaster. Every day he took a mysterious book out of his desk and absorbed himself in it at times when no classes were reciting. He kept that book under lock and key. There was not an urchin in school but was perishing to have a glimpse of it, but the chance never came. Every boy and girl had a theory about the nature of that book; but no two theories were alike, and there was no way of getting at the facts in

the case. Now, as Becky was passing by the desk, which stood near the door, she noticed that the key was in the lock! It was a precious moment. She glanced around; found herself alone, and the next instant she had the book in her hands. The title-page -- Professor Somebody's Anatomy -- carried no information to her mind; so she began to turn the leaves. She came at once upon a handsomely engraved and colored frontispiece -- a human figure, stark naked. At that moment a shadow fell on the page and Tom Sawyer stepped in at the door and caught a glimpse of the picture. Becky snatched at the book to close it, and had the hard luck to tear the pictured page half down the middle. She thrust the volume into the desk, turned the key, and burst out crying with shame and vexation.

"Tom Sawyer, you are just as mean as you can be, to sneak up on a person and look at what they're looking at."

"How could I know you was looking at anything?"

"You ought to be ashamed of yourself, Tom Sawyer; you know you're going to tell on me, and oh, what shall I do, what shall I do! I'll be whipped, and I never was whipped in school."

Then she stamped her little foot and said:

"Be so mean if you want to! I know something that's going to happen. You just wait and you'll see! Hateful, hateful, hateful!" -- and she flung out of the house with a new explosion of crying.

Tom stood still, rather flustered by this onslaught. Presently he said to himself:

"What a curious kind of a fool a girl is! Never been licked in school! Shucks! What's a licking! That's just like a girl -- they're so thin-skinned and chicken-hearted. Well, of course I ain't going to tell old Dobbins on this little fool, because there's other ways of getting even on her, that ain't so mean; but what of it? Old Dobbins will ask who it was tore his book. Nobody'll answer. Then he'll do just the way he always does -- ask first one and then t'other, and when he comes to the right girl he'll know it, without any telling. Girls' faces always tell on them. They ain't got any backbone. She'll get licked. Well, it's a kind of a tight place for Becky Thatcher, because there ain't any way out of it." Tom conned the thing a moment longer, and then added: "All right, though; she'd like to see me in just such a fix -- let her sweat it out!"

Tom joined the mob of skylarking scholars outside. In a few moments the master arrived and school "took in." Tom did not feel a strong interest in his studies. Every time he stole a glance at the girls' side of the room Becky's face troubled him. Considering all things, he did not want to pity her, and yet it was all he could do to help it. He could get up no exultation that was really worthy the name. Presently the spelling-book discovery was made, and Tom's mind was entirely full of his own matters for a while after that. Becky roused up from her lethargy of distress and showed good interest in the proceedings. She did not expect that Tom could get out of his trouble by denying that he spilt the ink on the book himself; and she was right. The denial only seemed to make the thing worse for Tom. Becky supposed she would be glad of that, and she tried to believe she was glad of it, but she found she was not certain. When the worst came to the worst, she had an impulse to get up and tell on Alfred Temple, but she made an effort and forced herself to keep still -- because, said she to herself, "he'll tell about me tearing the picture sure. I wouldn't say a word, not to save his life!"

Tom took his whipping and went back to his seat not at all broken-hearted, for he thought it was possible that he had unknowingly upset the ink on the spelling-book himself, in some skylarking bout -- he had denied it for form's sake and because it was custom, and had stuck to the denial from principle.

A whole hour drifted by, the master sat nodding in his throne, the air was drowsy with the hum of study. By and by, Mr. Dobbins straightened himself up, yawned, then unlocked his desk, and reached for his book, but seemed undecided whether to take it out or leave it. Most of the pupils glanced up languidly, but there were two among them that watched his movements with intent eyes. Mr. Dobbins fingered his book absently for a while, then took it out and settled himself in his chair to read! Tom shot a glance at Becky. He had seen a hunted and helpless rabbit look as she did, with a gun levelled at its head. Instantly he forgot his quarrel with her. Quick -- something must be done! done in a flash, too! But the very imminence of the emergency paralyzed his invention. Good! -- he had an inspiration! He would run and snatch the book, spring through the door and fly. But his resolution shook for one

little instant, and the chance was lost -- the master opened the volume. If Tom only had the wasted opportunity back again! Too late. There was no help for Becky now, he said. The next moment the master faced the school. Every eye sank under his gaze. There was that in it which smote even the innocent with fear. There was silence while one might count ten -- the master was gathering his wrath. Then he spoke: "Who tore this book?"

There was not a sound. One could have heard a pin drop. The stillness continued; the master searched face after face for signs of guilt.

"Benjamin Rogers, did you tear this book?"

A denial. Another pause.

"Joseph Harper, did you?"

Another denial. Tom's uneasiness grew more and more intense under the slow torture of these proceedings. The master scanned the ranks of boys -- considered a while, then turned to the girls:

"Amy Lawrence?"

A shake of the head.

"Gracie Miller?"

The same sign.

"Susan Harper, did you do this?"

Another negative. The next girl was Becky Thatcher. Tom was trembling from head to foot with excitement and a sense of the hopelessness of the situation.

"Rebecca Thatcher" [Tom glanced at her face -- it was white with terror] -- "did you tear -- no, look me in the face" [her hands rose in appeal] -- "did you tear this book?"

A thought shot like lightning through Tom's brain. He sprang to his feet and shouted -- "I done it!"

The school stared in perplexity at this incredible folly. Tom stood a moment, to gather his dismembered faculties; and when he stepped forward to go to his punishment the surprise, the gratitude, the adoration that shone upon him out of poor Becky's eyes seemed pay enough for a hundred floggings. Inspired by the splendor of his own act, he took without an outcry the most merciless flogging that even Mr. Dobbins had ever administered; and also received with indifference the added cruelty of a command to remain two hours after school should be dismissed -- for he knew who would wait for him outside till his captivity was done, and not count the tedious time as loss, either.

Tom went to bed that night planning vengeance against Alfred Temple; for with shame and repentance Becky had told him all, not forgetting her own treachery; but even the longing for vengeance had to give way, soon, to pleasanter musings, and he fell asleep at last with Becky's latest words lingering dreamily in his ear --

"Tom, how could you be so noble!"

The Adventures of Tom Sawyer/Chapter XXI

Vacation was approaching. The schoolmaster, always severe, grew severer and more exacting than ever, for he wanted the school to make a good showing on "Examination" day. His rod and his ferule were seldom idle now -- at least among the smaller pupils. Only the biggest boys, and young ladies of eighteen and twenty, escaped lashing. Mr. Dobbins' lashings were very vigorous ones, too; for although he carried, under his wig, a perfectly bald and shiny head, he had only reached middle age, and there was no sign of feebleness in his muscle. As the great day approached, all the tyranny that was in him came to the surface; he seemed to take a vindictive pleasure in punishing the least shortcomings. The consequence was, that the smaller boys spent their days in terror and suffering and their nights in plotting revenge. They threw away no opportunity to do the master a mischief. But he kept ahead all the time. The retribution that followed every vengeful success was so sweeping and majestic that the boys always retired from the field badly worsted. At last they conspired together and hit upon a plan that promised a dazzling victory. They swore in the sign-painter's boy, told him the scheme, and asked his help. He had his own reasons for being delighted, for the master boarded in his father's family and had given the boy ample cause to hate him. The master's wife would go on a visit to the country in a few days, and there would be nothing to interfere with the plan; the master always prepared himself for great occasions by getting pretty well fuddled, and the sign-painter's boy said that when the dominie had reached the proper condition on Examination Evening he would "manage the thing" while he napped in his chair; then he would have him awakened at the right time and hurried away to school. In the fulness of time the interesting occasion arrived. At eight in the evening the schoolhouse was brilliantly lighted, and adorned with wreaths and festoons of foliage and flowers. The master sat throned in his great chair upon a raised platform, with his blackboard behind him. He was looking tolerably mellow. Three rows of benches on each side and six rows in front of him were occupied by the dignitaries of the town and by the parents of the pupils. To his left, back of the rows of citizens, was a spacious temporary platform upon which were seated the scholars who were to take part in the exercises of the evening; rows of small boys, washed and dressed to an intolerable state of discomfort; rows of gawky big boys; snowbanks of girls and young ladies clad in lawn and muslin and conspicuously conscious of their bare arms, their grandmothers' ancient trinkets, their bits of pink and blue ribbon and the flowers in their hair. All the rest of the house was filled with non-participating scholars.

The exercises began. A very little boy stood up and sheepishly recited, "You'd scarce expect one of my age to speak in public on the stage," etc. -- accompanying himself with the painfully exact and spasmodic gestures which a machine might have used -- supposing the machine to be a trifle out of order. But he got through safely, though cruelly scared, and got a fine round of applause when he made his manufactured bow and retired.

A little shamefaced girl lisped, "Mary had a little lamb," etc., performed a compassion-inspiring curtsy, got her meed of applause, and sat down flushed and happy.

Tom Sawyer stepped forward with conceited confidence and soared into the unquenchable and indestructible "Give me liberty or give me death" speech, with fine fury and frantic gesticulation, and broke down in the middle of it. A ghastly stage-fright seized him, his legs quaked under him and he was like to choke. True, he had the manifest sympathy of the house but he had the house's silence, too, which was even worse than its sympathy. The master frowned, and this completed the disaster. Tom struggled awhile and then retired, utterly defeated. There was a weak attempt at applause, but it died early.

"The Boy Stood on the Burning Deck" followed; also "The Assyrian Came Down," and other declamatory gems. Then there were reading exercises, and a spelling fight. The meagre Latin class recited with honor. The prime feature of the evening was in order, now -- original "compositions" by the young ladies. Each in her turn stepped forward to the edge of the platform, cleared her throat, held up her manuscript (tied with dainty ribbon), and proceeded to read, with labored attention to "expression" and punctuation. The themes were the same that had been illuminated upon similar occasions by their mothers before them, their grandmothers, and doubtless all their ancestors in the female line clear back to the Crusades. "Friendship" was one; "Memories of Other Days"; "Religion in History"; "Dream

Land"; "The Advantages of Culture"; "Forms of Political Government Compared and Contrasted"; "Melancholy"; "Filial Love"; "Heart Longings," etc., etc.

A prevalent feature in these compositions was a nursed and petted melancholy; another was a wasteful and opulent gush of "fine language"; another was a tendency to lug in by the ears particularly prized words and phrases until they were worn entirely out; and a peculiarity that conspicuously marked and marred them was the inveterate and intolerable sermon that wagged its crippled tail at the end of each and every one of them. No matter what the subject might be, a brain-racking effort was made to squirm it into some aspect or other that the moral and religious mind could contemplate with edification. The glaring insincerity of these sermons was not sufficient to compass the banishment of the fashion from the schools, and it is not sufficient to-day; it never will be sufficient while the world stands, perhaps. There is no school in all our land where the young ladies do not feel obliged to close their compositions with a sermon; and you will find that the sermon of the most frivolous and the least religious girl in the school is always the longest and the most relentlessly pious. But enough of this. Homely truth is unpalatable.

Let us return to the "Examination." The first composition that was read was one entitled "Is this, then, Life?" Perhaps the reader can endure an extract from it:

"In the common walks of life, with what delightful emotions does the youthful mind look forward to some anticipated scene of festivity! Imagination is busy sketching rose-tinted pictures of joy. In fancy, the voluptuous votary of fashion sees herself amid the festive throng, 'the observed of all observers.' Her graceful form, arrayed in snowy robes, is whirling through the mazes of the joyous dance; her eye is brightest, her step is lightest in the gay assembly. "In such delicious fancies time quickly glides by, and the welcome hour arrives for her entrance into the Elysian world, of which she has had such bright dreams. How fairy-like does everything appear to her enchanted vision! Each new scene is more charming than the last. But after a while she finds that beneath this goodly exterior, all is vanity, the flattery which once charmed her soul, now grates harshly upon her ear; the ball-room has lost its charms; and with wasted health and imbittered heart, she turns away with the conviction that earthly pleasures cannot satisfy the longings of the soul!"

And so forth and so on. There was a buzz of gratification from time to time during the reading, accompanied by whispered ejaculations of "How sweet!" "How eloquent!" "So true!" etc., and after the thing had closed with a peculiarly afflicting sermon the applause was enthusiastic. Then arose a slim, melancholy girl, whose face had the "interesting" paleness that comes of pills and indigestion, and read a "poem." Two stanzas of it will do:

"A Missouri Maiden's Farewell to Alabama

"Alabama, good-bye! I love thee well!
But yet for a while do I leave thee now!
Sad, yes, sad thoughts of thee my heart doth swell,
And burning recollections throng my brow!
For I have wandered through thy flowery woods;
Have roamed and read near Tallapoosa's stream;
Have listened to Tallassee's warring floods,
And wooed on Coosa's side Aurora's beam.
"Yet shame I not to bear an o'er-full heart,
Nor blush to turn behind my tearful eyes;
'Tis from no stranger land I now must part,
'Tis to no strangers left I yield these sighs.
Welcome and home were mine within this State,
Whose vales I leave -- whose spires fade fast from me
And cold must be mine eyes, and heart, and tete,
When, dear Alabama! they turn cold on thee!"

There were very few there who knew what "tete" meant, but the poem was very satisfactory, nevertheless. Next appeared a dark-complexioned, black-eyed, black-haired young lady, who paused an impressive moment, assumed a tragic expression, and began to read in a measured, solemn tone:

"A Vision "Dark and tempestuous was night. Around the throne on high not a single star quivered; but the deep intonations of the heavy thunder constantly vibrated upon the ear; whilst the terrific lightning revelled in angry mood through the cloudy chambers of heaven, seeming to scorn the power exerted over its terror by the illustrious Franklin! Even the boisterous winds unanimously came forth from their mystic homes, and blustered about as if to enhance by their aid the wildness of the scene.

"At such a time,so dark,so dreary, for human sympathy my very spirit sighed; but instead thereof,

"My dearest friend, my counsellor, my comforter and guide -- My joy in grief, my second bliss in joy,' came to my side. She moved like one of those bright beings pictured in the sunny walks of fancy's Eden by the romantic and young, a queen of beauty unadorned save by her own transcendent loveliness. So soft was her step, it failed to make even a sound, and but for the magical thrill imparted by her genial touch, as other unobtrusive beauties, she would have glided away un-perceived -- unsought. A strange sadness rested upon her features, like icy tears upon the robe of December, as she pointed to the contending elements without, and bade me contemplate the two beings presented."

This nightmare occupied some ten pages of manuscript and wound up with a sermon so destructive of all hope to non-Presbyterians that it took the first prize. This composition was considered to be the very finest effort of the evening. The mayor of the village, in delivering the prize to the author of it, made a warm speech in which he said that it was by far the most "eloquent" thing he had ever listened to, and that Daniel Webster himself might well be proud of it. It may be remarked, in passing, that the number of compositions in which the word "beauteous" was over-fondled, and human experience referred to as "life's page," was up to the usual average.

Now the master, mellow almost to the verge of geniality, put his chair aside, turned his back to the audience, and began to draw a map of America on the blackboard, to exercise the geography class upon. But he made a sad business of it with his unsteady hand, and a smothered titter rippled over the house. He knew what the matter was, and set himself to right it. He sponged out lines and remade them; but he only distorted them more than ever, and the tittering was more pronounced. He threw his entire attention upon his work, now, as if determined not to be put down by the mirth. He felt that all eyes were fastened upon him; he imagined he was succeeding, and yet the tittering continued; it even manifestly increased. And well it might. There was a garret above, pierced with a scuttle over his head; and down through this scuttle came a cat, suspended around the haunches by a string; she had a rag tied about her head and jaws to keep her from mewling; as she slowly descended she curved upward and clawed at the string, she swung downward and clawed at the intangible air. The tittering rose higher and higher -- the cat was within six inches of the absorbed teacher's head -- down, down, a little lower, and she grabbed his wig with her desperate claws, clung to it, and was snatched up into the garret in an instant with her trophy still in her possession! And how the light did blaze abroad from the master's bald pate -- for the sign-painter's boy had gilded it!

That broke up the meeting. The boys were avenged. Vacation had come.

NOTE:-- The pretended "compositions" quoted in this chapter are taken without alteration from a volume entitled "Prose and Poetry, by a Western Lady" -- but they are exactly and precisely after the schoolgirl pattern, and hence are much happier than any mere imitations could be.

The Adventures of Tom Sawyer/Chapter XXII

Tom joined the new order of Cadets of Temperance, being attracted by the showy character of their "regalia." He promised to abstain from smoking, chewing, and profanity as long as he remained a member. Now he found out a new thing -- namely, that to promise not to do a thing is the surest way in the world to make a body want to go and do that very thing. Tom soon found himself tormented with a desire to drink and swear; the desire grew to be so intense that nothing but the hope of a chance to display himself in his red sash kept him from withdrawing from the order. Fourth of July was coming; but he soon gave that up -- gave it up before he had worn his shackles over forty-eight hours -- and fixed his hopes upon old Judge Frazer, justice of the peace, who was apparently on his deathbed and would have a big public funeral, since he was so high an official. During three days Tom was deeply concerned about the Judge's condition and hungry for news of it. Sometimes his hopes ran high -- so high that he would venture to get out his regalia and practise before the looking-glass. But the Judge had a most discouraging way of fluctuating. At last he was pronounced upon the mend -- and then convalescent. Tom was disgusted; and felt a sense of injury, too. He handed in his resignation at once -- and that night the Judge suffered a relapse and died. Tom resolved that he would never trust a man like that again. The funeral was a fine thing. The Cadets paraded in a style calculated to kill the late member with envy. Tom was a free boy again, however -- there was something in that. He could drink and swear, now -- but found to his surprise that he did not want to. The simple fact that he could, took the desire away, and the charm of it.

Tom presently wondered to find that his coveted vacation was beginning to hang a little heavily on his hands.

He attempted a diary -- but nothing happened during three days, and so he abandoned it.

The first of all the negro minstrel shows came to town, and made a sensation. Tom and Joe Harper got up a band of performers and were happy for two days.

Even the Glorious Fourth was in some sense a failure, for it rained hard, there was no procession in consequence, and the greatest man in the world (as Tom supposed), Mr. Benton, an actual United States Senator, proved an overwhelming disappointment -- for he was not twenty-five feet high, nor even anywhere in the neighborhood of it.

A circus came. The boys played circus for three days afterward in tents made of rag carpeting -- admission, three pins for boys, two for girls -- and then circusing was abandoned.

A phrenologist and a mesmerizer came -- and went again and left the village duller and drearier than ever.

There were some boys-and-girls' parties, but they were so few and so delightful that they only made the aching voids between ache the harder.

Becky Thatcher was gone to her Constantinople home to stay with her parents during vacation -- so there was no bright side to life anywhere.

The dreadful secret of the murder was a chronic misery. It was a very cancer for permanency and pain.

Then came the measles.

During two long weeks Tom lay a prisoner, dead to the world and its happenings. He was very ill, he was interested in nothing. When he got upon his feet at last and moved feebly down-town, a melancholy change had come over everything and every creature. There had been a "revival," and everybody had "got religion," not only the adults, but even the boys and girls. Tom went about, hoping against hope for the sight of one blessed sinful face, but disappointment crossed him everywhere. He found Joe Harper studying a Testament, and turned sadly away from the depressing spectacle. He sought Ben Rogers, and found him visiting the poor with a basket of tracts. He hunted up Jim Hollis, who called his attention to the precious blessing of his late measles as a warning. Every boy he encountered added another ton to his depression; and when, in desperation, he flew for refuge at last to the bosom of Huckleberry Finn and was received with a Scriptural quotation, his heart broke and he crept home and to bed realizing that he alone of all the town was lost, forever and forever.

And that night there came on a terrific storm, with driving rain, awful claps of thunder and blinding sheets of lightning. He covered his head with the bedclothes and waited in a horror of suspense for his doom; for he had not the shadow of a doubt that all this hubbub was about him. He believed he had taxed the forbearance of the powers above to the extremity of endurance and that this was the result. It might have seemed to him a waste of pomp and ammunition to kill a bug with a battery of artillery, but there seemed nothing incongruous about the getting up such an expensive thunderstorm as this to knock the turf from under an insect like himself.

By and by the tempest spent itself and died without accomplishing its object. The boy's first impulse was to be grateful, and reform. His second was to wait -- for there might not be any more storms.

The next day the doctors were back; Tom had relapsed. The three weeks he spent on his back this time seemed an entire age. When he got abroad at last he was hardly grateful that he had been spared, remembering how lonely was his estate, how companionless and forlorn he was. He drifted listlessly down the street and found Jim Hollis acting as judge in a juvenile court that was trying a cat for murder, in the presence of her victim, a bird. He found Joe Harper and Huck Finn up an alley eating a stolen melon. Poor lads! they -- like Tom -- had suffered a relapse.

The Adventures of Tom Sawyer/Chapter XXIII

At last the sleepy atmosphere was stirred -- and vigorously: the murder trial came on in the court. It became the absorbing topic of village talk immediately. Tom could not get away from it. Every reference to the murder sent a shudder to his heart, for his troubled conscience and fears almost persuaded him that these remarks were put forth in his hearing as "feelers"; he did not see how he could be suspected of knowing anything about the murder, but still he could not be comfortable in the midst of this gossip. It kept him in a cold shiver all the time. He took Huck to a lonely place to have a talk with him. It would be some relief to unseal his tongue for a little while; to divide his burden of distress with another sufferer. Moreover, he wanted to assure himself that Huck had remained discreet. "Huck, have you ever told anybody about -- that?"

"Bout what?"

"You know what."

"Oh -- 'course I haven't."

"Never a word?"

"Never a solitary word, so help me. What makes you ask?"

"Well, I was afeard."

"Why, Tom Sawyer, we wouldn't be alive two days if that got found out. You know that."

Tom felt more comfortable. After a pause:

"Huck, they couldn't anybody get you to tell, could they?"

"Get me to tell? Why, if I wanted that half-breed devil to drownd me they could get me to tell. They ain't no different way."

"Well, that's all right, then. I reckon we're safe as long as we keep mum. But let's swear again, anyway. It's more surer."

"I'm agreed."

So they swore again with dread solemnities.

"What is the talk around, Huck? I've heard a power of it."

"Talk? Well, it's just Muff Potter, Muff Potter, Muff Potter all the time. It keeps me in a sweat, constant, so's I want to hide som'ers."

"That's just the same way they go on round me. I reckon he's a goner. Don't you feel sorry for him, sometimes?"

"Most always -- most always. He ain't no account; but then he hain't ever done anything to hurt anybody. Just fishes a little, to get money to get drunk on -- and loaf around considerable; but lord, we all do that -- leastways most of us -- preachers and such like. But he's kind of good -- he give me half a fish, once, when there warn't enough for two; and lots of times he's kind of stood by me when I was out of luck."

"Well, he's mended kites for me, Huck, and knitted hooks on to my line. I wish we could get him out of there."

"My! we couldn't get him out, Tom. And besides, 'twouldn't do any good; they'd ketch him again."

"Yes -- so they would. But I hate to hear 'em abuse him so like the dickens when he never done -- that."

"I do too, Tom. Lord, I hear 'em say he's the bloodiest looking villain in this country, and they wonder he wasn't ever hung before."

"Yes, they talk like that, all the time. I've heard 'em say that if he was to get free they'd lynch him."

"And they'd do it, too."

The boys had a long talk, but it brought them little comfort. As the twilight drew on, they found themselves hanging about the neighborhood of the little isolated jail, perhaps with an undefined hope that something would happen that might clear away their difficulties. But nothing happened; there seemed to be no angels or fairies interested in this luckless captive.

The boys did as they had often done before -- went to the cell grating and gave Potter some tobacco and matches. He was on the ground floor and there were no guards.

His gratitude for their gifts had always smote their consciences before -- it cut deeper than ever, this time. They felt cowardly and treacherous to the last degree when Potter said:

"You've been mighty good to me, boys -- better'n anybody else in this town. And I don't forget it, I don't. Often I says to myself, says I, 'I used to mend all the boys' kites and things, and show 'em where the good fishin' places was, and befriend 'em what I could, and now they've all forgot old Muff when he's in trouble; but Tom don't, and Huck don't -- they don't forget him, says I, 'and I don't forget them.' Well, boys, I done an awful thing -- drunk and crazy at the time -- that's the only way I account for it -- and now I got to swing for it, and it's right. Right, and best, too, I reckon -- hope so, anyway. Well, we won't talk about that. I don't want to make you feel bad; you've befriended me. But what I want to say, is, don't you ever get drunk -- then you won't ever get here. Stand a litter furdur west -- so -- that's it; it's a prime comfort to see faces that's friendly when a body's in such a muck of trouble, and there don't none come here but yourn. Good friendly faces -- good friendly faces. Git up on one another's backs and let me touch 'em. That's it. Shake hands -- yourn'll come through the bars, but mine's too big. Little hands, and weak -- but they've helped Muff Potter a power, and they'd help him more if they could."

Tom went home miserable, and his dreams that night were full of horrors. The next day and the day after, he hung about the court-room, drawn by an almost irresistible impulse to go in, but forcing himself to stay out. Huck was having the same experience. They studiously avoided each other. Each wandered away, from time to time, but the same dismal fascination always brought them back presently. Tom kept his ears open when idlers sauntered out of the courtroom, but invariably heard distressing news -- the toils were closing more and more relentlessly around poor Potter. At the end of the second day the village talk was to the effect that Injun Joe's evidence stood firm and unshaken, and that there was not the slightest question as to what the jury's verdict would be.

Tom was out late, that night, and came to bed through the window. He was in a tremendous state of excitement. It was hours before he got to sleep. All the village flocked to the court-house the next morning, for this was to be the great day. Both sexes were about equally represented in the packed audience. After a long wait the jury filed in and took their places; shortly afterward, Potter, pale and haggard, timid and hopeless, was brought in, with chains upon him, and seated where all the curious eyes could stare at him; no less conspicuous was Injun Joe, stolid as ever. There was another pause, and then the judge arrived and the sheriff proclaimed the opening of the court. The usual whisperings among the lawyers and gathering together of papers followed. These details and accompanying delays worked up an atmosphere of preparation that was as impressive as it was fascinating.

Now a witness was called who testified that he found Muff Potter washing in the brook, at an early hour of the morning that the murder was discovered, and that he immediately sneaked away. After some further questioning, counsel for the prosecution said:

"Take the witness."

The prisoner raised his eyes for a moment, but dropped them again when his own counsel said:

"I have no questions to ask him."

The next witness proved the finding of the knife near the corpse. Counsel for the prosecution said:

"Take the witness."

"I have no questions to ask him," Potter's lawyer replied.

A third witness swore he had often seen the knife in Potter's possession.

"Take the witness."

Counsel for Potter declined to question him. The faces of the audience began to betray annoyance. Did this attorney mean to throw away his client's life without an effort?

Several witnesses deposed concerning Potter's guilty behavior when brought to the scene of the murder. They were allowed to leave the stand without being cross-questioned.

Every detail of the damaging circumstances that occurred in the graveyard upon that morning which all present remembered so well was brought out by credible witnesses, but none of them were cross-examined by Potter's lawyer. The perplexity and dissatisfaction of the house expressed itself in murmurs and provoked a reproof from the bench. Counsel for the prosecution now said:

"By the oaths of citizens whose simple word is above suspicion, we have fastened this awful crime, beyond all possibility of question, upon the unhappy prisoner at the bar. We rest our case here."

A groan escaped from poor Potter, and he put his face in his hands and rocked his body softly to and fro, while a painful silence reigned in the court-room. Many men were moved, and many women's compassion testified itself in tears. Counsel for the defence rose and said:

"Your honor, in our remarks at the opening of this trial, we foreshadowed our purpose to prove that our client did this fearful deed while under the influence of a blind and irresponsible delirium produced by drink. We have changed our mind. We shall not offer that plea." [Then to the clerk:] "Call Thomas Sawyer!"

A puzzled amazement awoke in every face in the house, not even excepting Potter's. Every eye fastened itself with wondering interest upon Tom as he rose and took his place upon the stand. The boy looked wild enough, for he was badly scared. The oath was administered.

"Thomas Sawyer, where were you on the seventeenth of June, about the hour of midnight?"

Tom glanced at Injun Joe's iron face and his tongue failed him. The audience listened breathless, but the words refused to come. After a few moments, however, the boy got a little of his strength back, and managed to put enough of it into his voice to make part of the house hear:

"In the graveyard!"

"A little bit louder, please. Don't be afraid. You were --"

"In the graveyard."

A contemptuous smile flitted across Injun Joe's face.

"Were you anywhere near Horse Williams' grave?"

"Yes, sir."

"Speak up -- just a trifle louder. How near were you?"

"Near as I am to you."

"Were you hidden, or not?"

"I was hid."

"Where?"

"Behind the elms that's on the edge of the grave."

Injun Joe gave a barely perceptible start.

"Any one with you?"

"Yes, sir. I went there with --"

"Wait -- wait a moment. Never mind mentioning your companion's name. We will produce him at the proper time. Did you carry anything there with you."

Tom hesitated and looked confused.

"Speak out, my boy -- don't be diffident. The truth is always respectable. What did you take there?"

"Only a -- a -- dead cat."

There was a ripple of mirth, which the court checked.

"We will produce the skeleton of that cat. Now, my boy, tell us everything that occurred -- tell it in your own way -- don't skip anything, and don't be afraid."

Tom began -- hesitatingly at first, but as he warmed to his subject his words flowed more and more easily; in a little while every sound ceased but his own voice; every eye fixed itself upon him; with parted lips and bated breath the audience hung upon his words, taking no note of time, rapt in the ghastly fascinations of the tale. The strain upon pent emotion reached its climax when the boy said:

"-- and as the doctor fetched the board around and Muff Potter fell, Injun Joe jumped with the knife and --"

Crash! Quick as lightning the half-breed sprang for a window, tore his way through all opposers, and was gone!

The Adventures of Tom Sawyer/Chapter XXIV

Tom was a glittering hero once more -- the pet of the old, the envy of the young. His name even went into immortal print, for the village paper magnified him. There were some that believed he would be President, yet, if he escaped hanging. As usual, the fickle, unreasoning world took Muff Potter to its bosom and fondled him as lavishly as it had abused him before. But that sort of conduct is to the world's credit; therefore it is not well to find fault with it.

Tom's days were days of splendor and exultation to him, but his nights were seasons of horror. Injun Joe infested all his dreams, and always with doom in his eye. Hardly any temptation could persuade the boy to stir abroad after nightfall. Poor Huck was in the same state of wretchedness and terror, for Tom had told the whole story to the lawyer the night before the great day of the trial, and Huck was sore afraid that his share in the business might leak out, yet, notwithstanding Injun Joe's flight had saved him the suffering of testifying in court. The poor fellow had got the attorney to promise secrecy, but what of that? Since Tom's harassed conscience had managed to drive him to the lawyer's house by night and wring a dread tale from lips that had been sealed with the dismalest and most formidable of oaths, Huck's confidence in the human race was well-nigh obliterated.

Daily Muff Potter's gratitude made Tom glad he had spoken; but nightly he wished he had sealed up his tongue.

Half the time Tom was afraid Injun Joe would never be captured; the other half he was afraid he would be. He felt sure he never could draw a safe breath again until that man was dead and he had seen the corpse.

Rewards had been offered, the country had been scoured, but no Injun Joe was found. One of those omniscient and awe-inspiring marvels, a detective, came up from St. Louis, moused around, shook his head, looked wise, and made that sort of astounding success which members of that craft usually achieve. That is to say, he "found a clew." But you can't hang a "clew" for murder, and so after that detective had got through and gone home, Tom felt just as insecure as he was before.

The slow days drifted on, and each left behind it a slightly lightened weight of apprehension.

The Adventures of Tom Sawyer/Chapter XXV

There comes a time in every rightly-constructed boy's life when he has a raging desire to go somewhere and dig for hidden treasure. This desire suddenly came upon Tom one day. He sallied out to find Joe Harper, but failed of success. Next he sought Ben Rogers; he had gone fishing. Presently he stumbled upon Huck Finn the Red-Handed. Huck would answer. Tom took him to a private place and opened the matter to him confidentially. Huck was willing. Huck was always willing to take a hand in any enterprise that offered entertainment and required no capital, for he had a troublesome superabundance of that sort of time which is not money. "Where'll we dig?" said Huck. "Oh, most anywhere."

"Why, is it hid all around?"

"No, indeed it ain't. It's hid in mighty particular places, Huck -- sometimes on islands, sometimes in rotten chests under the end of a limb of an old dead tree, just where the shadow falls at midnight; but mostly under the floor in ha'nted houses."

"Who hides it?"

"Why, robbers, of course -- who'd you reckon? Sunday-school sup'rintendents?"

"I don't know. If 'twas mine I wouldn't hide it; I'd spend it and have a good time."

"So would I. But robbers don't do that way. They always hide it and leave it there."

"Don't they come after it any more?"

"No, they think they will, but they generally forget the marks, or else they die. Anyway, it lays there a long time and gets rusty; and by and by somebody finds an old yellow paper that tells how to find the marks -- a paper that's got to be ciphered over about a week because it's mostly signs and hy'roglyphics."

"HyroQwhich?"

"Hy'roglyphics -- pictures and things, you know, that don't seem to mean anything."

"Have you got one of them papers, Tom?"

"No."

"Well then, how you going to find the marks?"

"I don't want any marks. They always bury it under a ha'nted house or on an island, or under a dead tree that's got one limb sticking out. Well, we've tried Jackson's Island a little, and we can try it again some time; and there's the old ha'nted house up the Still-House branch, and there's lots of deadlimb trees -- dead loads of 'em."

"Is it under all of them?"

"How you talk! No!"

"Then how you going to know which one to go for?"

"Go for all of 'em!"

"Why, Tom, it'll take all summer."

"Well, what of that? Suppose you find a brass pot with a hundred dollars in it, all rusty and gray, or rotten chest full of di'monds. How's that?"

Huck's eyes glowed.

"That's bully. Plenty bully enough for me. Just you gimme the hundred dollars and I don't want no di'monds."

"All right. But I bet you I ain't going to throw off on di'monds. Some of 'em's worth twenty dollars apiece -- there ain't any, hardly, but's worth six bits or a dollar."

"No! Is that so?"

"Cert'nly -- anybody'll tell you so. Hain't you ever seen one, Huck?"

"Not as I remember."

"Oh, kings have slathers of them."

"Well, I don' know no kings, Tom."

"I reckon you don't. But if you was to go to Europe you'd see a raft of 'em hopping around."

"Do they hop?"

"Hop? -- your granny! No!"

"Well, what did you say they did, for?"

"Shucks, I only meant you'd see 'em -- not hopping, of course -- what do they want to hop for? -- but I mean you'd just see 'em -- scattered around, you know, in a kind of a general way. Like that old humpbacked Richard."

"Richard? What's his other name?"

"He didn't have any other name. Kings don't have any but a given name."

"No?"

"But they don't."

"Well, if they like it, Tom, all right; but I don't want to be a king and have only just a given name, like a nigger. But say -- where you going to dig first?"

"Well, I don't know. S'pose we tackle that old dead-limb tree on the hill t'other side of Still-House branch?"

"I'm agreed."

So they got a crippled pick and a shovel, and set out on their three-mile tramp. They arrived hot and panting, and threw themselves down in the shade of a neighboring elm to rest and have a smoke.

"I like this," said Tom.

"So do I."

"Say, Huck, if we find a treasure here, what you going to do with your share?"

"Well, I'll have pie and a glass of soda every day, and I'll go to every circus that comes along. I bet I'll have a gay time."

"Well, ain't you going to save any of it?"

"Save it? What for?"

"Why, so as to have something to live on, by and by."

"Oh, that ain't any use. Pap would come back to thish-er town some day and get his claws on it if I didn't hurry up, and I tell you he'd clean it out pretty quick. What you going to do with yourn, Tom?"

"I'm going to buy a new drum, and a sure-'nough sword, and a red necktie and a bull pup, and get married."

"Married!"

"That's it."

"Tom, you -- why, you ain't in your right mind."

"Wait -- you'll see."

"Well, that's the foolishhest thing you could do. Look at pap and my mother. Fight! Why, they used to fight all the time. I remember, mighty well."

"That ain't anything. The girl I'm going to marry won't fight."

"Tom, I reckon they're all alike. They'll all comb a body. Now you better think 'bout this awhile. I tell you you better. What's the name of the gal?"

"It ain't a gal at all -- it's a girl."

"It's all the same, I reckon; some says gal, some says girl -- both's right, like enough. Anyway, what's her name, Tom?"

"I'll tell you some time -- not now."

"All right -- that'll do. Only if you get married I'll be more lonesomer than ever."

"No you won't. You'll come and live with me. Now stir out of this and we'll go to digging."

They worked and sweated for half an hour. No result. They toiled another half-hour. Still no result. Huck said:

"Do they always bury it as deep as this?"

"Sometimes -- not always. Not generally. I reckon we haven't got the right place."

So they chose a new spot and began again. The labor dragged a little, but still they made progress. They pegged away in silence for some time. Finally Huck leaned on his shovel, swabbed the beaded drops from his brow with his sleeve, and said:

"Where you going to dig next, after we get this one?"

"I reckon maybe we'll tackle the old tree that's over yonder on Cardiff Hill back of the widow's."

"I reckon that'll be a good one. But won't the widow take it away from us, Tom? It's on her land."

"She take it away! Maybe she'd like to try it once. Whoever finds one of these hid treasures, it belongs to him. It don't make any difference whose land it's on."

That was satisfactory. The work went on. By and by Huck said:

"Blame it, we must be in the wrong place again. What do you think?"

"It is mighty curious, Huck. I don't understand it. Sometimes witches interfere. I reckon maybe that's what's the trouble now."

"Shucks! Witches ain't got no power in the daytime."

"Well, that's so. I didn't think of that. Oh, I know what the matter is! What a blamed lot of fools we are! You got to find out where the shadow of the limb falls at midnight, and that's where you dig!"

"Then consound it, we've fooled away all this work for nothing. Now hang it all, we got to come back in the night. It's an awful long way. Can you get out?"

"I bet I will. We've got to do it to-night, too, because if somebody sees these holes they'll know in a minute what's here and they'll go for it."

"Well, I'll come around and maow to-night."

"All right. Let's hide the tools in the bushes."

The boys were there that night, about the appointed time. They sat in the shadow waiting. It was a lonely place, and an hour made solemn by old traditions. Spirits whispered in the rustling leaves, ghosts lurked in the murky nooks, the deep baying of a hound floated up out of the distance, an owl answered with his sepulchral note. The boys were subdued by these solemnities, and talked little. By and by they judged that twelve had come; they marked where the shadow fell, and began to dig. Their hopes commenced to rise. Their interest grew stronger, and their industry kept pace with it. The hole deepened and still deepened, but every time their hearts jumped to hear the pick strike upon something, they only suffered a new disappointment. It was only a stone or a chunk. At last Tom said:

"It ain't any use, Huck, we're wrong again."

"Well, but we can't be wrong. We spotted the shadder to a dot."

"I know it, but then there's another thing."

"What's that?"

"Why, we only guessed at the time. Like enough it was too late or too early."

Huck dropped his shovel.

"That's it," said he. "That's the very trouble. We got to give this one up. We can't ever tell the right time, and besides this kind of thing's too awful, here this time of night with witches and ghosts a-fluttering around so. I feel as if something's behind me all the time; and I'm afeard to turn around, becuz maybe there's others in front a-waiting for a chance. I been creeping all over, ever since I got here."

"Well, I've been pretty much so, too, Huck. They most always put in a dead man when they bury a treasure under a tree, to look out for it."

"Lordy!"

"Yes, they do. I've always heard that."

"Tom, I don't like to fool around much where there's dead people. A body's bound to get into trouble with 'em, sure."

"I don't like to stir 'em up, either. S'pose this one here was to stick his skull out and say something!"

"Don't Tom! It's awful."

"Well, it just is. Huck, I don't feel comfortable a bit."

"Say, Tom, let's give this place up, and try somewheres else."

"All right, I reckon we better."

"What'll it be?"

Tom considered awhile; and then said:

"The ha'nted house. That's it!"

"Blame it, I don't like ha'nted houses, Tom. Why, they're a dern sight worse'n dead people. Dead people might talk, maybe, but they don't come sliding around in a shroud, when you ain't noticing, and peep over your shoulder all of a sudden and grit their teeth, the way a ghost does. I couldn't stand such a thing as that, Tom -- nobody could."

"Yes, but, Huck, ghosts don't travel around only at night. They won't hender us from digging there in the daytime."

"Well, that's so. But you know mighty well people don't go about that ha'nted house in the day nor the night."

"Well, that's mostly because they don't like to go where a man's been murdered, anyway -- but nothing's ever been seen around that house except in the night -- just some blue lights slipping by the windows -- no regular ghosts."

"Well, where you see one of them blue lights flickering around, Tom, you can bet there's a ghost mighty close behind it. It stands to reason. Becuz you know that they don't anybody but ghosts use 'em."

"Yes, that's so. But anyway they don't come around in the daytime, so what's the use of our being afeard?"

"Well, all right. We'll tackle the ha'nted house if you say so -- but I reckon it's taking chances."

They had started down the hill by this time. There in the middle of the moonlit valley below them stood the "ha'nted" house, utterly isolated, its fences gone long ago, rank weeds smothering the very doorsteps, the chimney crumbled to ruin, the window-sashes vacant, a corner of the roof caved in. The boys gazed awhile, half expecting to see a blue light flit past a window; then talking in a low tone, as befitted the time and the circumstances, they struck far off to the right, to give the haunted house a wide berth, and took their way homeward through the woods that adorned the rearward side of Cardiff Hill.

The Adventures of Tom Sawyer/Chapter XXVI

About noon the next day the boys arrived at the dead tree; they had come for their tools. Tom was impatient to go to the haunted house; Huck was measurably so, also -- but suddenly said: "Lookyhere, Tom, do you know what day it is?"

Tom mentally ran over the days of the week, and then quickly lifted his eyes with a startled look in them --

"My! I never once thought of it, Huck!"

"Well, I didn't neither, but all at once it popped onto me that it was Friday."

"Blame it, a body can't be too careful, Huck. We might 'a' got into an awful scrape, tackling such a thing on a Friday."

"Might! Better say we would! There's some lucky days, maybe, but Friday ain't."

"Any fool knows that. I don't reckon you was the first that found it out, Huck."

"Well, I never said I was, did I? And Friday ain't all, neither. I had a rotten bad dream last night -- dreamt about rats."

"No! Sure sign of trouble. Did they fight?"

"No."

"Well, that's good, Huck. When they don't fight it's only a sign that there's trouble around, you know. All we got to do is to look mighty sharp and keep out of it. We'll drop this thing for to-day, and play. Do you know Robin Hood, Huck?"

"No. Who's Robin Hood?"

"Why, he was one of the greatest men that was ever in England -- and the best. He was a robber."

"Cracky, I wisht I was. Who did he rob?"

"Only sheriffs and bishops and rich people and kings, and such like. But he never bothered the poor. He loved 'em. He always divided up with 'em perfectly square."

"Well, he must 'a' been a brick."

"I bet you he was, Huck. Oh, he was the noblest man that ever was. They ain't any such men now, I can tell you. He could lick any man in England, with one hand tied behind him; and he could take his yew bow and plug a ten-cent piece every time, a mile and a half."

"What's a yew bow?"

"I don't know. It's some kind of a bow, of course. And if he hit that dime only on the edge he would set down and cry -- and curse. But we'll play Robin Hood -- it's nobby fun. I'll learn you."

"I'm agreed."

So they played Robin Hood all the afternoon, now and then casting a yearning eye down upon the haunted house and passing a remark about the morrow's prospects and possibilities there. As the sun began to sink into the west they took their way homeward athwart the long shadows of the trees and soon were buried from sight in the forests of Cardiff Hill.

On Saturday, shortly after noon, the boys were at the dead tree again. They had a smoke and a chat in the shade, and then dug a little in their last hole, not with great hope, but merely because Tom said there were so many cases where people had given up a treasure after getting down within six inches of it, and then somebody else had come along and turned it up with a single thrust of a shovel. The thing failed this time, however, so the boys shouldered their tools and went away feeling that they had not trifled with fortune, but had fulfilled all the requirements that belong to the business of treasure-hunting.

When they reached the haunted house there was something so weird and grisly about the dead silence that reigned there under the baking sun, and something so depressing about the loneliness and desolation of the place, that they were afraid, for a moment, to venture in. Then they crept to the door and took a trembling peep. They saw a weed-grown, floorless room, unplastered, an ancient fireplace, vacant windows, a ruinous staircase; and here, there, and everywhere hung ragged and abandoned cobwebs. They presently entered, softly, with quickened pulses, talking in whispers, ears alert to catch the slightest sound, and muscles tense and ready for instant retreat.

In a little while familiarity modified their fears and they gave the place a critical and interested examination, rather admiring their own boldness, and wondering at it, too. Next they wanted to look up-stairs. This was something like cutting off retreat, but they got to daring each other, and of course there could be but one result -- they threw their tools into a corner and made the ascent. Up there were the same signs of decay. In one corner they found a closet that promised mystery, but the promise was a fraud -- there was nothing in it. Their courage was up now and well in hand. They were about to go down and begin work when --

"Sh!" said Tom.

"What is it?" whispered Huck, blanching with fright.

"Sh! ... There! ... Hear it?"

"Yes! ... Oh, my! Let's run!"

"Keep still! Don't you budge! They're coming right toward the door."

The boys stretched themselves upon the floor with their eyes to knot-holes in the planking, and lay waiting, in a misery of fear.

"They've stopped.... No -- coming.... Here they are. Don't whisper another word, Huck. My goodness, I wish I was out of this!"

Two men entered. Each boy said to himself: "There's the old deaf and dumb Spaniard that's been about town once or twice lately -- never saw t'other man before."

"T'other" was a ragged, unkempt creature, with nothing very pleasant in his face. The Spaniard was wrapped in a serape; he had bushy white whiskers; long white hair flowed from under his sombrero, and he wore green goggles. When they came in, "t'other" was talking in a low voice; they sat down on the ground, facing the door, with their backs to the wall, and the speaker continued his remarks. His manner became less guarded and his words more distinct as he proceeded:

"No," said he, "I've thought it all over, and I don't like it. It's dangerous."

"Dangerous!" grunted the "deaf and dumb" Spaniard -- to the vast surprise of the boys. "Milk-sop!"

This voice made the boys gasp and quake. It was Injun Joe's! There was silence for some time. Then Joe said:

"What's any more dangerous than that job up yonder -- but nothing's come of it."

"That's different. Away up the river so, and not another house about. 'Twon't ever be known that we tried, anyway, long as we didn't succeed."

"Well, what's more dangerous than coming here in the daytime! -- anybody would suspicion us that saw us."

"I know that. But there warn't any other place as handy after that fool of a job. I want to quit this shanty. I wanted to yesterday, only it warn't any use trying to stir out of here, with those infernal boys playing over there on the hill right in full view."

"Those infernal boys" quaked again under the inspiration of this remark, and thought how lucky it was that they had remembered it was Friday and concluded to wait a day. They wished in their hearts they had waited a year.

The two men got out some food and made a luncheon. After a long and thoughtful silence, Injun Joe said:

"Look here, lad -- you go back up the river where you belong. Wait there till you hear from me. I'll take the chances on dropping into this town just once more, for a look. We'll do that 'dangerous' job after I've spied around a little and

think things look well for it. Then for Texas! We'll leg it together!"

This was satisfactory. Both men presently fell to yawning, and Injun Joe said:

"I'm dead for sleep! It's your turn to watch."

He curled down in the weeds and soon began to snore. His comrade stirred him once or twice and he became quiet.

Presently the watcher began to nod; his head drooped lower and lower, both men began to snore now.

The boys drew a long, grateful breath. Tom whispered:

"Now's our chance -- come!"

Huck said:

"I can't -- I'd die if they was to wake."

Tom urged -- Huck held back. At last Tom rose slowly and softly, and started alone. But the first step he made wrung such a hideous creak from the crazy floor that he sank down almost dead with fright. He never made a second attempt. The boys lay there counting the dragging moments till it seemed to them that time must be done and eternity growing gray; and then they were grateful to note that at last the sun was setting.

Now one snore ceased. Injun Joe sat up, stared around -- smiled grimly upon his comrade, whose head was drooping upon his knees -- stirred him up with his foot and said:

"Here! You're a watchman, ain't you! All right, though -- nothing's happened."

"My! have I been asleep?"

"Oh, partly, partly. Nearly time for us to be moving, pard. What'll we do with what little swag we've got left?"

"I don't know -- leave it here as we've always done, I reckon. No use to take it away till we start south. Six hundred and fifty in silver's something to carry."

"Well -- all right -- it won't matter to come here once more."

"No -- but I'd say come in the night as we used to do -- it's better."

"Yes; but look here; it may be a good while before I get the right chance at that job; accidents might happen; 'tain't in such a very good place; we'll just regularly bury it -- and bury it deep."

"Good idea," said the comrade, who walked across the room, knelt down, raised one of the rearward hearthstones and took out a bag that jingled pleasantly. He subtracted from it twenty or thirty dollars for himself and as much for Injun Joe, and passed the bag to the latter, who was on his knees in the corner, now, digging with his bowie-knife.

The boys forgot all their fears, all their miseries in an instant. With gloating eyes they watched every movement. Luck! -- the splendor of it was beyond all imagination! Six hundred dollars was money enough to make half a dozen boys rich! Here was treasure-hunting under the happiest auspices -- there would not be any bothersome uncertainty as to where to dig. They nudged each other every moment -- eloquent nudges and easily understood, for they simply meant -- "Oh, but ain't you glad now we're here!"

Joe's knife struck upon something.

"Hello!" said he.

"What is it?" said his comrade.

"Half-rotten plank -- no, it's a box, I believe. Here -- bear a hand and we'll see what it's here for. Never mind, I've broke a hole."

He reached his hand in and drew it out --

"Man, it's money!"

The two men examined the handful of coins. They were gold. The boys above were as excited as themselves, and as delighted.

Joe's comrade said:

"We'll make quick work of this. There's an old rusty pick over amongst the weeds in the corner the other side of the fireplace -- I saw it a minute ago."

He ran and brought the boys' pick and shovel. Injun Joe took the pick, looked it over critically, shook his head, muttered something to himself, and then began to use it. The box was soon unearthed. It was not very large; it was iron bound and had been very strong before the slow years had injured it. The men contemplated the treasure awhile in blissful silence.

"Pard, there's thousands of dollars here," said Injun Joe.

"'Twas always said that Murrel's gang used to be around here one summer," the stranger observed.

"I know it," said Injun Joe; "and this looks like it, I should say."

"Now you won't need to do that job."

The half-breed frowned. Said he:

"You don't know me. Least you don't know all about that thing. 'Tain't robbery altogether -- it's revenge!" and a wicked light flamed in his eyes. "I'll need your help in it. When it's finished -- then Texas. Go home to your Nance and your kids, and stand by till you hear from me."

"Well -- if you say so; what'll we do with this -- bury it again?"

"Yes. [Ravishing delight overhead.] No! by the great Sachem, no! [Profound distress overhead.] I'd nearly forgot. That pick had fresh earth on it! [The boys were sick with terror in a moment.] What business has a pick and a shovel here? What business with fresh earth on them? Who brought them here -- and where are they gone? Have you heard anybody? -- seen anybody? What! bury it again and leave them to come and see the ground disturbed? Not exactly -- not exactly. We'll take it to my den."

"Why, of course! Might have thought of that before. You mean Number One?"

"No -- Number Two -- under the cross. The other place is bad -- too common."

"All right. It's nearly dark enough to start."

Injun Joe got up and went about from window to window cautiously peeping out. Presently he said:

"Who could have brought those tools here? Do you reckon they can be up-stairs?"

The boys' breath forsook them. Injun Joe put his hand on his knife, halted a moment, undecided, and then turned toward the stairway. The boys thought of the closet, but their strength was gone. The steps came creaking up the stairs -- the intolerable distress of the situation woke the stricken resolution of the lads -- they were about to spring for the closet, when there was a crash of rotten timbers and Injun Joe landed on the ground amid the debris of the ruined stairway. He gathered himself up cursing, and his comrade said:

"Now what's the use of all that? If it's anybody, and they're up there, let them stay there -- who cares? If they want to jump down, now, and get into trouble, who objects? It will be dark in fifteen minutes -- and then let them follow us if they want to. I'm willing. In my opinion, whoever hove those things in here caught a sight of us and took us for ghosts or devils or something. I'll bet they're running yet."

Joe grumbled awhile; then he agreed with his friend that what daylight was left ought to be economized in getting things ready for leaving. Shortly afterward they slipped out of the house in the deepening twilight, and moved toward the river with their precious box.

Tom and Huck rose up, weak but vastly relieved, and stared after them through the chinks between the logs of the house. Follow? Not they. They were content to reach ground again without broken necks, and take the townward track over the hill. They did not talk much. They were too much absorbed in hating themselves -- hating the ill luck that made them take the spade and the pick there. But for that, Injun Joe never would have suspected. He would have hidden the silver with the gold to wait there till his "revenge" was satisfied, and then he would have had the misfortune to find that money turn up missing. Bitter, bitter luck that the tools were ever brought there!

They resolved to keep a lookout for that Spaniard when he should come to town spying out for chances to do his revengeful job, and follow him to "Number Two," wherever that might be. Then a ghastly thought occurred to Tom.

"Revenge? What if he means us, Huck!"

"Oh, don't!" said Huck, nearly fainting.

They talked it all over, and as they entered town they agreed to believe that he might possibly mean somebody else -- at least that he might at least mean nobody but Tom, since only Tom had testified.

Very, very small comfort it was to Tom to be alone in danger! Company would be a palpable improvement, he thought.

The Adventures of Tom Sawyer/Chapter XXVII

The adventure of the day mightily tormented Tom's dreams that night. Four times he had his hands on that rich treasure and four times it wasted to nothingness in his fingers as sleep forsook him and wakefulness brought back the hard reality of his misfortune. As he lay in the early morning recalling the incidents of his great adventure, he noticed that they seemed curiously subdued and far away -- somewhat as if they had happened in another world, or in a time long gone by. Then it occurred to him that the great adventure itself must be a dream! There was one very strong argument in favor of this idea -- namely, that the quantity of coin he had seen was too vast to be real. He had never seen as much as fifty dollars in one mass before, and he was like all boys of his age and station in life, in that he imagined that all references to "hundreds" and "thousands" were mere fanciful forms of speech, and that no such sums really existed in the world. He never had supposed for a moment that so large a sum as a hundred dollars was to be found in actual money in any one's possession. If his notions of hidden treasure had been analyzed, they would have been found to consist of a handful of real dimes and a bushel of vague, splendid, ungraspable dollars.

But the incidents of his adventure grew sensibly sharper and clearer under the attrition of thinking them over, and so he presently found himself leaning to the impression that the thing might not have been a dream, after all. This uncertainty must be swept away. He would snatch a hurried breakfast and go and find Huck. Huck was sitting on the gunwale of a flatboat, listlessly dangling his feet in the water and looking very melancholy. Tom concluded to let Huck lead up to the subject. If he did not do it, then the adventure would be proved to have been only a dream.

"Hello, Huck!"

"Hello, yourself."

Silence, for a minute.

"Tom, if we'd 'a' left the blame tools at the dead tree, we'd 'a' got the money. Oh, ain't it awful!"

"'Tain't a dream, then, 'tain't a dream! Somehow I most wish it was. Dog'd if I don't, Huck."

"What ain't a dream?"

"Oh, that thing yesterday. I been half thinking it was."

"Dream! If them stairs hadn't broke down you'd 'a' seen how much dream it was! I've had dreams enough all night -- with that patch-eyed Spanish devil going for me all through 'em -- rot him!"

"No, not rot him. Find him! Track the money!"

"Tom, we'll never find him. A feller don't have only one chance for such a pile -- and that one's lost. I'd feel mighty shaky if I was to see him, anyway."

"Well, so'd I; but I'd like to see him, anyway -- and track him out -- to his Number Two."

"Number Two -- yes, that's it. I been thinking 'bout that. But I can't make nothing out of it. What do you reckon it is?"

"I dono. It's too deep. Say, Huck -- maybe it's the number of a house!"

"Goody! ... No, Tom, that ain't it. If it is, it ain't in this one-horse town. They ain't no numbers here."

"Well, that's so. Lemme think a minute. Here -- it's the number of a room -- in a tavern, you know!"

"Oh, that's the trick! They ain't only two taverns. We can find out quick."

"You stay here, Huck, till I come."

Tom was off at once. He did not care to have Huck's company in public places. He was gone half an hour. He found that in the best tavern, No. 2 had long been occupied by a young lawyer, and was still so occupied. In the less ostentatious house, No. 2 was a mystery. The tavern-keeper's young son said it was kept locked all the time, and he never saw anybody go into it or come out of it except at night; he did not know any particular reason for this state of things; had had some little curiosity, but it was rather feeble; had made the most of the mystery by entertaining himself with the idea that that room was "ha'nted"; had noticed that there was a light in there the night before.

"That's what I've found out, Huck. I reckon that's the very No. 2 we're after."

"I reckon it is, Tom. Now what you going to do?"

"Lemme think."

Tom thought a long time. Then he said:

"I'll tell you. The back door of that No. 2 is the door that comes out into that little close alley between the tavern and the old rattle trap of a brick store. Now you get hold of all the door-keys you can find, and I'll nip all of auntie's, and the first dark night we'll go there and try 'em. And mind you, keep a lookout for Injun Joe, because he said he was going to drop into town and spy around once more for a chance to get his revenge. If you see him, you just follow him; and if he don't go to that No. 2, that ain't the place."

"Lordy, I don't want to foller him by myself!"

"Why, it'll be night, sure. He mightn't ever see you -- and if he did, maybe he'd never think anything."

"Well, if it's pretty dark I reckon I'll track him. I dono -- I dono. I'll try."

"You bet I'll follow him, if it's dark, Huck. Why, he might 'a' found out he couldn't get his revenge, and be going right after that money."

"It's so, Tom, it's so. I'll foller him; I will, by jingoes!"

"Now you're talking! Don't you ever weaken, Huck, and I won't."

The Adventures of Tom Sawyer/Chapter XXVIII

That night Tom and Huck were ready for their adventure. They hung about the neighborhood of the tavern until after nine, one watching the alley at a distance and the other the tavern door. Nobody entered the alley or left it; nobody resembling the Spaniard entered or left the tavern door. The night promised to be a fair one; so Tom went home with the understanding that if a considerable degree of darkness came on, Huck was to come and "maow," whereupon he would slip out and try the keys. But the night remained clear, and Huck closed his watch and retired to bed in an empty sugar hogshead about twelve. Tuesday the boys had the same ill luck. Also Wednesday. But Thursday night promised better. Tom slipped out in good season with his aunt's old tin lantern, and a large towel to blindfold it with. He hid the lantern in Huck's sugar hogshead and the watch began. An hour before midnight the tavern closed up and its lights (the only ones thereabouts) were put out. No Spaniard had been seen. Nobody had entered or left the alley. Everything was auspicious. The blackness of darkness reigned, the perfect stillness was interrupted only by occasional mutterings of distant thunder.

Tom got his lantern, lit it in the hogshead, wrapped it closely in the towel, and the two adventurers crept in the gloom toward the tavern. Huck stood sentry and Tom felt his way into the alley. Then there was a season of waiting anxiety that weighed upon Huck's spirits like a mountain. He began to wish he could see a flash from the lantern -- it would frighten him, but it would at least tell him that Tom was alive yet. It seemed hours since Tom had disappeared. Surely he must have fainted; maybe he was dead; maybe his heart had burst under terror and excitement. In his uneasiness Huck found himself drawing closer and closer to the alley; fearing all sorts of dreadful things, and momentarily expecting some catastrophe to happen that would take away his breath. There was not much to take away, for he seemed only able to inhale it by thimblefuls, and his heart would soon wear itself out, the way it was beating. Suddenly there was a flash of light and Tom came tearing by him:

.

"Run!" said he; "run, for your life!"

He needn't have repeated it; once was enough; Huck was making thirty or forty miles an hour before the repetition was uttered. The boys never stopped till they reached the shed of a deserted slaughter-house at the lower end of the village. Just as they got within its shelter the storm burst and the rain poured down. As soon as Tom got his breath he said:

"Huck, it was awful! I tried two of the keys, just as soft as I could; but they seemed to make such a power of racket that I couldn't hardly get my breath I was so scared. They wouldn't turn in the lock, either. Well, without noticing what I was doing, I took hold of the knob, and open comes the door! It warn't locked! I hopped in, and shook off the towel, and, great Caesar's ghost!"

"What! -- what'd you see, Tom?"

"Huck, I most stepped onto Injun Joe's hand!"

"No!"

"Yes! He was lying there, sound asleep on the floor, with his old patch on his eye and his arms spread out."

"Lordy, what did you do? Did he wake up?"

"No, never budged. Drunk, I reckon. I just grabbed that towel and started!"

"I'd never 'a' thought of the towel, I bet!"

"Well, I would. My aunt would make me mighty sick if I lost it."

"Say, Tom, did you see that box?"

"Huck, I didn't wait to look around. I didn't see the box, I didn't see the cross. I didn't see anything but a bottle and a tin cup on the floor by Injun Joe; yes, I saw two barrels and lots more bottles in the room. Don't you see, now, what's the matter with that ha'nted room?"

"How?"

"Why, it's ha'nted with whiskey! Maybe all the Temperance Taverns have got a ha'nted room, hey, Huck?"

"Well, I reckon maybe that's so. Who'd 'a' thought such a thing? But say, Tom, now's a mighty good time to get that box, if Injun Joe's drunk."

"It is, that! You try it!"

Huck shuddered.

"Well, no -- I reckon not."

"And I reckon not, Huck. Only one bottle alongside of Injun Joe ain't enough. If there'd been three, he'd be drunk enough and I'd do it."

There was a long pause for reflection, and then Tom said:

"Lookyhere, Huck, less not try that thing any more till we know Injun Joe's not in there. It's too scary. Now, if we watch every night, we'll be dead sure to see him go out, some time or other, and then we'll snatch that box quicker'n lightning."

"Well, I'm agreed. I'll watch the whole night long, and I'll do it every night, too, if you'll do the other part of the job."

"All right, I will. All you got to do is to trot up Hooper Street a block and maow -- and if I'm asleep, you throw some gravel at the window and that'll fetch me."

"Agreed, and good as wheat!"

"Now, Huck, the storm's over, and I'll go home. It'll begin to be daylight in a couple of hours. You go back and watch that long, will you?"

"I said I would, Tom, and I will. I'll ha'nt that tavern every night for a year! I'll sleep all day and I'll stand watch all night."

"That's all right. Now, where you going to sleep?"

"In Ben Rogers' hayloft. He lets me, and so does his pap's nigger man, Uncle Jake. I tote water for Uncle Jake whenever he wants me to, and any time I ask him he gives me a little something to eat if he can spare it. That's a mighty good nigger, Tom. He likes me, becuz I don't ever act as if I was above him. Sometime I've set right down and eat with him. But you needn't tell that. A body's got to do things when he's awful hungry he wouldn't want to do as a steady thing."

"Well, if I don't want you in the daytime, I'll let you sleep. I won't come bothering around. Any time you see something's up, in the night, just skip right around and maow."

The Adventures of Tom Sawyer/Chapter XXIX

The first thing Tom heard on Friday morning was a glad piece of news -- Judge Thatcher's family had come back to town the night before. Both Injun Joe and the treasure sunk into secondary importance for a moment, and Becky took the chief place in the boy's interest. He saw her and they had an exhausting good time playing "hi-spy" and "gully-keeper" with a crowd of their schoolmates. The day was completed and crowned in a peculiarly satisfactory way: Becky teased her mother to appoint the next day for the long-promised and long-delayed picnic, and she consented. The child's delight was boundless; and Tom's not more moderate. The invitations were sent out before sunset, and straightway the young folks of the village were thrown into a fever of preparation and pleasurable anticipation. Tom's excitement enabled him to keep awake until a pretty late hour, and he had good hopes of hearing Huck's "maow," and of having his treasure to astonish Becky and the picnickers with, next day; but he was disappointed. No signal came that night. Morning came, eventually, and by ten or eleven o'clock a giddy and rollicking company were gathered at Judge Thatcher's, and everything was ready for a start. It was not the custom for elderly people to mar the picnics with their presence. The children were considered safe enough under the wings of a few young ladies of eighteen and a few young gentlemen of twenty-three or thereabouts. The old steam ferryboat was chartered for the occasion; presently the gay throng filed up the main street laden with provision-baskets. Sid was sick and had to miss the fun; Mary remained at home to entertain him. The last thing Mrs. Thatcher said to Becky, was:

"You'll not get back till late. Perhaps you'd better stay all night with some of the girls that live near the ferry-landing, child."

"Then I'll stay with Susy Harper, mamma."

"Very well. And mind and behave yourself and don't be any trouble."

Presently, as they tripped along, Tom said to Becky:

"Say -- I'll tell you what we'll do. 'Stead of going to Joe Harper's we'll climb right up the hill and stop at the Widow Douglas'. She'll have ice-cream! She has it most every day -- dead loads of it. And she'll be awful glad to have us."

"Oh, that will be fun!"

Then Becky reflected a moment and said:

"But what will mamma say?"

"How'll she ever know?"

The girl turned the idea over in her mind, and said reluctantly:

"I reckon it's wrong -- but --"

"But shucks! Your mother won't know, and so what's the harm? All she wants is that you'll be safe; and I bet you she'd 'a' said go there if she'd 'a' thought of it. I know she would!"

The Widow Douglas' splendid hospitality was a tempting bait. It and Tom's persuasions presently carried the day. So it was decided to say nothing anybody about the night's programme. Presently it occurred to Tom that maybe Huck might come this very night and give the signal. The thought took a deal of the spirit out of his anticipations. Still he could not bear to give up the fun at Widow Douglas'. And why should he give it up, he reasoned -- the signal did not come the night before, so why should it be any more likely to come to-night? The sure fun of the evening outweighed the uncertain treasure; and, boylike, he determined to yield to the stronger inclination and not allow himself to think of the box of money another time that day.

Three miles below town the ferryboat stopped at the mouth of a woody hollow and tied up. The crowd swarmed ashore and soon the forest distances and craggy heights echoed far and near with shoutings and laughter. All the different ways of getting hot and tired were gone through with, and by-and-by the rovers straggled back to camp fortified with responsible appetites, and then the destruction of the good things began. After the feast there was a

refreshing season of rest and chat in the shade of spreading oaks. By-and-by somebody shouted:

"Who's ready for the cave?"

Everybody was. Bundles of candles were procured, and straightway there was a general scamper up the hill. The mouth of the cave was up the hillside -- an opening shaped like a letter A. Its massive oaken door stood unbarred. Within was a small chamber, chilly as an ice-house, and walled by Nature with solid limestone that was dewy with a cold sweat. It was romantic and mysterious to stand here in the deep gloom and look out upon the green valley shining in the sun. But the impressiveness of the situation quickly wore off, and the romping began again. The moment a candle was lighted there was a general rush upon the owner of it; a struggle and a gallant defence followed, but the candle was soon knocked down or blown out, and then there was a glad clamor of laughter and a new chase. But all things have an end. By-and-by the procession went filing down the steep descent of the main avenue, the flickering rank of lights dimly revealing the lofty walls of rock almost to their point of junction sixty feet overhead. This main avenue was not more than eight or ten feet wide. Every few steps other lofty and still narrower crevices branched from it on either hand -- for McDougal's cave was but a vast labyrinth of crooked aisles that ran into each other and out again and led nowhere. It was said that one might wander days and nights together through its intricate tangle of rifts and chasms, and never find the end of the cave; and that he might go down, and down, and still down, into the earth, and it was just the same -- labyrinth under labyrinth, and no end to any of them. No man "knew" the cave. That was an impossible thing. Most of the young men knew a portion of it, and it was not customary to venture much beyond this known portion. Tom Sawyer knew as much of the cave as any one.

The procession moved along the main avenue some three-quarters of a mile, and then groups and couples began to slip aside into branch avenues, fly along the dismal corridors, and take each other by surprise at points where the corridors joined again. Parties were able to elude each other for the space of half an hour without going beyond the "known" ground.

By-and-by, one group after another came straggling back to the mouth of the cave, panting, hilarious, smeared from head to foot with tallow drippings, daubed with clay, and entirely delighted with the success of the day. Then they were astonished to find that they had been taking no note of time and that night was about at hand. The clanging bell had been calling for half an hour. However, this sort of close to the day's adventures was romantic and therefore satisfactory. When the ferryboat with her wild freight pushed into the stream, nobody cared sixpence for the wasted time but the captain of the craft.

Huck was already upon his watch when the ferryboat's lights went glinting past the wharf. He heard no noise on board, for the young people were as subdued and still as people usually are who are nearly tired to death. He wondered what boat it was, and why she did not stop at the wharf -- and then he dropped her out of his mind and put his attention upon his business. The night was growing cloudy and dark. Ten o'clock came, and the noise of vehicles ceased, scattered lights began to wink out, all straggling foot-passengers disappeared, the village betook itself to its slumbers and left the small watcher alone with the silence and the ghosts. Eleven o'clock came, and the tavern lights were put out; darkness everywhere, now. Huck waited what seemed a weary long time, but nothing happened. His faith was weakening. Was there any use? Was there really any use? Why not give it up and turn in?

A noise fell upon his ear. He was all attention in an instant. The alley door closed softly. He sprang to the corner of the brick store. The next moment two men brushed by him, and one seemed to have something under his arm. It must be that box! So they were going to remove the treasure. Why call Tom now? It would be absurd -- the men would get away with the box and never be found again. No, he would stick to their wake and follow them; he would trust to the darkness for security from discovery. So communing with himself, Huck stepped out and glided along behind the men, cat-like, with bare feet, allowing them to keep just far enough ahead not to be invisible.

They moved up the river street three blocks, then turned to the left up a cross-street. They went straight ahead, then, until they came to the path that led up Cardiff Hill; this they took. They passed by the old Welshman's house, half-way up the hill, without hesitating, and still climbed upward. Good, thought Huck, they will bury it in the old quarry. But they never stopped at the quarry. They passed on, up the summit. They plunged into the narrow path

between the tall sumach bushes, and were at once hidden in the gloom. Huck closed up and shortened his distance, now, for they would never be able to see him. He trotted along awhile; then slackened his pace, fearing he was gaining too fast; moved on a piece, then stopped altogether; listened; no sound; none, save that he seemed to hear the beating of his own heart. The hooting of an owl came over the hill -- ominous sound! But no footsteps. Heavens, was everything lost! He was about to spring with winged feet, when a man cleared his throat not four feet from him! Huck's heart shot into his throat, but he swallowed it again; and then he stood there shaking as if a dozen agues had taken charge of him at once, and so weak that he thought he must surely fall to the ground. He knew where he was. He knew he was within five steps of the stile leading into Widow Douglas' grounds. Very well, he thought, let them bury it there; it won't be hard to find.

Now there was a voice -- a very low voice -- Injun Joe's:

"Damn her, maybe she's got company -- there's lights, late as it is."

"I can't see any."

This was that stranger's voice -- the stranger of the haunted house. A deadly chill went to Huck's heart -- this, then, was the "revenge" job! His thought was, to fly. Then he remembered that the Widow Douglas had been kind to him more than once, and maybe these men were going to murder her. He wished he dared venture to warn her; but he knew he didn't dare -- they might come and catch him. He thought all this and more in the moment that elapsed between the stranger's remark and Injun Joe's next -- which was --

"Because the bush is in your way. Now -- this way -- now you see, don't you?"

"Yes. Well, there is company there, I reckon. Better give it up."

"Give it up, and I just leaving this country forever! Give it up and maybe never have another chance. I tell you again, as I've told you before, I don't care for her swag -- you may have it. But her husband was rough on me -- many times he was rough on me -- and mainly he was the justice of the peace that jugged me for a vagrant. And that ain't all. It ain't a millionth part of it! He had me horsewhipped! -- horsewhipped in front of the jail, like a nigger! -- with all the town looking on! Horsewhipped! -- do you understand? He took advantage of me and died. But I'll take it out of her."

"Oh, don't kill her! Don't do that!"

"Kill? Who said anything about killing? I would kill him if he was here; but not her. When you want to get revenge on a woman you don't kill her -- bosh! you go for her looks. You slit her nostrils -- you notch her ears like a sow!"

"By God, that's --"

"Keep your opinion to yourself! It will be safest for you. I'll tie her to the bed. If she bleeds to death, is that my fault? I'll not cry, if she does. My friend, you'll help me in this thing -- for my sake -- that's why you're here -- I mightn't be able alone. If you flinch, I'll kill you. Do you understand that? And if I have to kill you, I'll kill her -- and then I reckon nobody'll ever know much about who done this business."

"Well, if it's got to be done, let's get at it. The quicker the better -- I'm all in a shiver."

"Do it now? And company there? Look here -- I'll get suspicious of you, first thing you know. No -- we'll wait till the lights are out -- there's no hurry."

Huck felt that a silence was going to ensue -- a thing still more awful than any amount of murderous talk; so he held his breath and stepped gingerly back; planted his foot carefully and firmly, after balancing, one-legged, in a precarious way and almost toppling over, first on one side and then on the other. He took another step back, with the same elaboration and the same risks; then another and another, and -- a twig snapped under his foot! His breath stopped and he listened. There was no sound -- the stillness was perfect. His gratitude was measureless. Now he turned in his tracks, between the walls of sumach bushes -- turned himself as carefully as if he were a ship -- and then stepped quickly but cautiously along. When he emerged at the quarry he felt secure, and so he picked up his nimble heels and flew. Down, down he sped, till he reached the Welshman's. He banged at the door, and presently

the heads of the old man and his two stalwart sons were thrust from windows.

"What's the row there? Who's banging? What do you want?"

"Let me in -- quick! I'll tell everything."

"Why, who are you?"

"Huckleberry Finn -- quick, let me in!"

"Huckleberry Finn, indeed! It ain't a name to open many doors, I judge! But let him in, lads, and let's see what's the trouble."

"Please don't ever tell I told you," were Huck's first words when he got in. "Please don't -- I'd be killed, sure -- but the widow's been good friends to me sometimes, and I want to tell -- I will tell if you'll promise you won't ever say it was me."

"By George, he has got something to tell, or he wouldn't act so!" exclaimed the old man; "out with it and nobody here'll ever tell, lad."

Three minutes later the old man and his sons, well armed, were up the hill, and just entering the sumach path on tiptoe, their weapons in their hands. Huck accompanied them no further. He hid behind a great bowlder and fell to listening. There was a lagging, anxious silence, and then all of a sudden there was an explosion of firearms and a cry. Huck waited for no particulars. He sprang away and sped down the hill as fast as his legs could carry him.

The Adventures of Tom Sawyer/Chapter XXX

As the earliest suspicion of dawn appeared on Sunday morning, Huck came groping up the hill and rapped gently at the old Welshman's door. The inmates were asleep, but it was a sleep that was set on a hair-trigger, on account of the exciting episode of the night. A call came from a window: "Who's there!"

Huck's scared voice answered in a low tone:

"Please let me in! It's only Huck Finn!"

"It's a name that can open this door night or day, lad! -- and welcome!"

These were strange words to the vagabond boy's ears, and the pleasantest he had ever heard. He could not recollect that the closing word had ever been applied in his case before. The door was quickly unlocked, and he entered. Huck was given a seat and the old man and his brace of tall sons speedily dressed themselves.

"Now, my boy, I hope you're good and hungry, because breakfast will be ready as soon as the sun's up, and we'll have a piping hot one, too -- make yourself easy about that! I and the boys hoped you'd turn up and stop here last night."

"I was awful scared," said Huck, "and I run. I took out when the pistols went off, and I didn't stop for three mile. I've come now becuz I wanted to know about it, you know; and I come before daylight becuz I didn't want to run across them devils, even if they was dead."

"Well, poor chap, you do look as if you'd had a hard night of it -- but there's a bed here for you when you've had your breakfast. No, they ain't dead, lad -- we are sorry enough for that. You see we knew right where to put our hands on them, by your description; so we crept along on tiptoe till we got within fifteen feet of them -- dark as a cellar that sumach path was -- and just then I found I was going to sneeze. It was the meanest kind of luck! I tried to keep it back, but no use -- 'twas bound to come, and it did come! I was in the lead with my pistol raised, and when the sneeze started those scoundrels a-rustling to get out of the path, I sung out, 'Fire boys!' and blazed away at the place where the rustling was. So did the boys. But they were off in a jiffy, those villains, and we after them, down through the woods. I judge we never touched them. They fired a shot apiece as they started, but their bullets whizzed by and didn't do us any harm. As soon as we lost the sound of their feet we quit chasing, and went down and stirred up the

constables. They got a posse together, and went off to guard the river bank, and as soon as it is light the sheriff and a gang are going to beat up the woods. My boys will be with them presently. I wish we had some sort of description of those rascals -- 'twould help a good deal. But you couldn't see what they were like, in the dark, lad, I suppose?"

"Oh yes; I saw them down-town and follered them."

"Splendid! Describe them -- describe them, my boy!"

"One's the old deaf and dumb Spaniard that's ben around here once or twice, and t'other's a mean-looking, ragged --"

"That's enough, lad, we know the men! Happened on them in the woods back of the widow's one day, and they slunk away. Off with you, boys, and tell the sheriff -- get your breakfast to-morrow morning!"

The Welshman's sons departed at once. As they were leaving the room Huck sprang up and exclaimed:

"Oh, please don't tell anybody it was me that blowed on them! Oh, please!"

"All right if you say it, Huck, but you ought to have the credit of what you did."

"Oh no, no! Please don't tell!"

When the young men were gone, the old Welshman said:

"They won't tell -- and I won't. But why don't you want it known?"

Huck would not explain, further than to say that he already knew too much about one of those men and would not have the man know that he knew anything against him for the whole world -- he would be killed for knowing it, sure.

The old man promised secrecy once more, and said:

"How did you come to follow these fellows, lad? Were they looking suspicious?"

Huck was silent while he framed a duly cautious reply. Then he said:

"Well, you see, I'm a kind of a hard lot, -- least everybody says so, and I don't see nothing agin it -- and sometimes I can't sleep much, on account of thinking about it and sort of trying to strike out a new way of doing. That was the way of it last night. I couldn't sleep, and so I come along up-street 'bout midnight, a-turning it all over, and when I got to that old shackly brick store by the Temperance Tavern, I backed up agin the wall to have another think. Well, just then along comes these two chaps slipping along close by me, with something under their arm, and I reckoned they'd stole it. One was a-smoking, and t'other one wanted a light; so they stopped right before me and the cigars lit up their faces and I see that the big one was the deaf and dumb Spaniard, by his white whiskers and the patch on his eye, and t'other one was a rusty, ragged-looking devil."

"Could you see the rags by the light of the cigars?"

This staggered Huck for a moment. Then he said:

"Well, I don't know -- but somehow it seems as if I did."

"Then they went on, and you --"

"Follered 'em -- yes. That was it. I wanted to see what was up -- they sneaked along so. I dogged 'em to the widder's stile, and stood in the dark and heard the ragged one beg for the widder, and the Spaniard swear he'd spile her looks just as I told you and your two --"

"What! The deaf and dumb man said all that!"

Huck had made another terrible mistake! He was trying his best to keep the old man from getting the faintest hint of who the Spaniard might be, and yet his tongue seemed determined to get him into trouble in spite of all he could do. He made several efforts to creep out of his scrape, but the old man's eye was upon him and he made blunder after blunder. Presently the Welshman said:

"My boy, don't be afraid of me. I wouldn't hurt a hair of your head for all the world. No -- I'd protect you -- I'd protect you. This Spaniard is not deaf and dumb; you've let that slip without intending it; you can't cover that up now. You know something about that Spaniard that you want to keep dark. Now trust me -- tell me what it is, and trust me -- I won't betray you."

Huck looked into the old man's honest eyes a moment, then bent over and whispered in his ear:

"Tain't a Spaniard -- it's Injun Joe!"

The Welshman almost jumped out of his chair. In a moment he said:

"It's all plain enough, now. When you talked about notching ears and slitting noses I judged that that was your own embellishment, because white men don't take that sort of revenge. But an Injun! That's a different matter altogether."

During breakfast the talk went on, and in the course of it the old man said that the last thing which he and his sons had done, before going to bed, was to get a lantern and examine the stile and its vicinity for marks of blood. They found none, but captured a bulky bundle of --

"Of what?"

If the words had been lightning they could not have leaped with a more stunning suddenness from Huck's blanched lips. His eyes were staring wide, now, and his breath suspended -- waiting for the answer. The Welshman started -- stared in return -- three seconds -- five seconds -- ten -- then replied:

"Of burglar's tools. Why, what's the matter with you?"

Huck sank back, panting gently, but deeply, unutterably grateful. The Welshman eyed him gravely, curiously -- and presently said:

"Yes, burglar's tools. That appears to relieve you a good deal. But what did give you that turn? What were you expecting we'd found?"

Huck was in a close place -- the inquiring eye was upon him -- he would have given anything for material for a plausible answer -- nothing suggested itself -- the inquiring eye was boring deeper and deeper -- a senseless reply offered -- there was no time to weigh it, so at a venture he uttered it -- feebly:

"Sunday-school books, maybe."

Poor Huck was too distressed to smile, but the old man laughed loud and joyously, shook up the details of his anatomy from head to foot, and ended by saying that such a laugh was money in a man's pocket, because it cut down the doctor's bill like everything. Then he added:

"Poor old chap, you're white and jaded -- you ain't well a bit -- no wonder you're a little flighty and off your balance. But you'll come out of it. Rest and sleep will fetch you out all right, I hope."

Huck was irritated to think he had been such a goose and betrayed such a suspicious excitement, for he had dropped the idea that the parcel brought from the tavern was the treasure, as soon as he had heard the talk at the widow's stile. He had only thought it was not the treasure, however -- he had not known that it wasn't -- and so the suggestion of a captured bundle was too much for his self-possession. But on the whole he felt glad the little episode had happened, for now he knew beyond all question that that bundle was not the bundle, and so his mind was at rest and exceedingly comfortable. In fact, everything seemed to be drifting just in the right direction, now; the treasure must be still in No. 2, the men would be captured and jailed that day, and he and Tom could seize the gold that night without any trouble or any fear of interruption.

Just as breakfast was completed there was a knock at the door. Huck jumped for a hiding-place, for he had no mind to be connected even remotely with the late event. The Welshman admitted several ladies and gentlemen, among them the Widow Douglas, and noticed that groups of citizens were climbing up the hill -- to stare at the stile. So the news had spread. The Welshman had to tell the story of the night to the visitors. The widow's gratitude for her preservation was outspoken.

"Don't say a word about it, madam. There's another that you're more beholden to than you are to me and my boys, maybe, but he don't allow me to tell his name. We wouldn't have been there but for him."

Of course this excited a curiosity so vast that it almost belittled the main matter -- but the Welshman allowed it to eat into the vitals of his visitors, and through them be transmitted to the whole town, for he refused to part with his secret. When all else had been learned, the widow said:

"I went to sleep reading in bed and slept straight through all that noise. Why didn't you come and wake me?"

"We judged it warn't worth while. Those fellows warn't likely to come again -- they hadn't any tools left to work with, and what was the use of waking you up and scaring you to death? My three negro men stood guard at your house all the rest of the night. They've just come back."

More visitors came, and the story had to be told and retold for a couple of hours more.

There was no Sabbath-school during day-school vacation, but everybody was early at church. The stirring event was well canvassed. News came that not a sign of the two villains had been yet discovered. When the sermon was finished, Judge Thatcher's wife dropped alongside of Mrs. Harper as she moved down the aisle with the crowd and said:

"Is my Becky going to sleep all day? I just expected she would be tired to death."

"Your Becky?"

"Yes," with a startled look -- "didn't she stay with you last night?"

"Why, no."

Mrs. Thatcher turned pale, and sank into a pew, just as Aunt Polly, talking briskly with a friend, passed by. Aunt Polly said:

"Good-morning, Mrs. Thatcher. Good-morning, Mrs. Harper. I've got a boy that's turned up missing. I reckon my Tom stayed at your house last night -- one of you. And now he's afraid to come to church. I've got to settle with him."

Mrs. Thatcher shook her head feebly and turned paler than ever.

"He didn't stay with us," said Mrs. Harper, beginning to look uneasy. A marked anxiety came into Aunt Polly's face.

"Joe Harper, have you seen my Tom this morning?"

"No'm."

"When did you see him last?"

Joe tried to remember, but was not sure he could say. The people had stopped moving out of church. Whispers passed along, and a boding uneasiness took possession of every countenance. Children were anxiously questioned, and young teachers. They all said they had not noticed whether Tom and Becky were on board the ferryboat on the homeward trip; it was dark; no one thought of inquiring if any one was missing. One young man finally blurted out his fear that they were still in the cave! Mrs. Thatcher swooned away. Aunt Polly fell to crying and wringing her hands.

The alarm swept from lip to lip, from group to group, from street to street, and within five minutes the bells were wildly clanging and the whole town was up! The Cardiff Hill episode sank into instant insignificance, the burglars were forgotten, horses were saddled, skiffs were manned, the ferryboat ordered out, and before the horror was half an hour old, two hundred men were pouring down highroad and river toward the cave.

All the long afternoon the village seemed empty and dead. Many women visited Aunt Polly and Mrs. Thatcher and tried to comfort them. They cried with them, too, and that was still better than words. All the tedious night the town waited for news; but when the morning dawned at last, all the word that came was, "Send more candles -- and send food." Mrs. Thatcher was almost crazed; and Aunt Polly, also. Judge Thatcher sent messages of hope and encouragement from the cave, but they conveyed no real cheer.

The old Welshman came home toward daylight, spattered with candle-grease, smeared with clay, and almost worn out. He found Huck still in the bed that had been provided for him, and delirious with fever. The physicians were all at the cave, so the Widow Douglas came and took charge of the patient. She said she would do her best by him, because, whether he was good, bad, or indifferent, he was the Lord's, and nothing that was the Lord's was a thing to be neglected. The Welshman said Huck had good spots in him, and the widow said:

"You can depend on it. That's the Lord's mark. He don't leave it off. He never does. Puts it somewhere on every creature that comes from his hands."

Early in the forenoon parties of jaded men began to straggle into the village, but the strongest of the citizens continued searching. All the news that could be gained was that remotenesses of the cavern were being ransacked that had never been visited before; that every corner and crevice was going to be thoroughly searched; that wherever one wandered through the maze of passages, lights were to be seen flitting hither and thither in the distance, and shoutings and pistol-shots sent their hollow reverberations to the ear down the sombre aisles. In one place, far from the section usually traversed by tourists, the names "Becky & Tom" had been found traced upon the rocky wall with candle-smoke, and near at hand a grease-soiled bit of ribbon. Mrs. Thatcher recognized the ribbon and cried over it. She said it was the last relic she should ever have of her child; and that no other memorial of her could ever be so precious, because this one parted latest from the living body before the awful death came. Some said that now and then, in the cave, a far-away speck of light would glimmer, and then a glorious shout would burst forth and a score of men go trooping down the echoing aisle -- and then a sickening disappointment always followed; the children were not there; it was only a searcher's light.

Three dreadful days and nights dragged their tedious hours along, and the village sank into a hopeless stupor. No one had heart for anything. The accidental discovery, just made, that the proprietor of the Temperance Tavern kept liquor on his premises, scarcely fluttered the public pulse, tremendous as the fact was. In a lucid interval, Huck feebly led up to the subject of taverns, and finally asked -- dimly dreading the worst -- if anything had been discovered at the Temperance Tavern since he had been ill.

"Yes," said the widow.

Huck started up in bed, wild-eyed:

"What? What was it?"

"Liquor! -- and the place has been shut up. Lie down, child -- what a turn you did give me!"

"Only tell me just one thing -- only just one -- please! Was it Tom Sawyer that found it?"

The widow burst into tears. "Hush, hush, child, hush! I've told you before, you must not talk. You are very, very sick!"

Then nothing but liquor had been found; there would have been a great powwow if it had been the gold. So the treasure was gone forever -- gone forever! But what could she be crying about? Curious that she should cry.

These thoughts worked their dim way through Huck's mind, and under the weariness they gave him he fell asleep. The widow said to herself:

"There -- he's asleep, poor wreck. Tom Sawyer find it! Pity but somebody could find Tom Sawyer! Ah, there ain't many left, now, that's got hope enough, or strength enough, either, to go on searching."

The Adventures of Tom Sawyer/Chapter XXXI

Now to return to Tom and Becky's share in the picnic. They tripped along the murky aisles with the rest of the company, visiting the familiar wonders of the cave -- wonders dubbed with rather overdescriptive names, such as "The Drawing-Room," "The Cathedral," "Aladdin's Palace," and so on. Presently the hide-and-seek frolicking began, and Tom and Becky engaged in it with zeal until the exertion began to grow a trifle wearisome; then they wandered down a sinuous avenue holding their candles aloft and reading the tangled web-work of names, dates, post-office addresses, and mottoes with which the rocky walls had been frescoed (in candle-smoke). Still drifting along and talking, they scarcely noticed that they were now in a part of the cave whose walls were not frescoed. They smoked their own names under an overhanging shelf and moved on. Presently they came to a place where a little stream of water, trickling over a ledge and carrying a limestone sediment with it, had, in the slow-dragging ages, formed a laced and ruffled Niagara in gleaming and imperishable stone. Tom squeezed his small body behind it in order to illuminate it for Becky's gratification. He found that it curtained a sort of steep natural stairway which was enclosed between narrow walls, and at once the ambition to be a discoverer seized him. Becky responded to his call, and they made a smoke-mark for future guidance, and started upon their quest. They wound this way and that, far down into the secret depths of the cave, made another mark, and branched off in search of novelties to tell the upper world about. In one place they found a spacious cavern, from whose ceiling depended a multitude of shining stalactites of the length and circumference of a man's leg; they walked all about it, wondering and admiring, and presently left it by one of the numerous passages that opened into it. This shortly brought them to a bewitching spring, whose basin was incrustated with a frostwork of glittering crystals; it was in the midst of a cavern whose walls were supported by many fantastic pillars which had been formed by the joining of great stalactites and stalagmites together, the result of the ceaseless water-drip of centuries. Under the roof vast knots of bats had packed themselves together, thousands in a bunch; the lights disturbed the creatures and they came flocking down by hundreds, squeaking and darting furiously at the candles. Tom knew their ways and the danger of this sort of conduct. He seized Becky's hand and hurried her into the first corridor that offered; and none too soon, for a bat struck Becky's light out with its wing while she was passing out of the cavern. The bats chased the children a good distance; but the fugitives plunged into every new passage that offered, and at last got rid of the perilous things. Tom found a subterranean lake, shortly, which stretched its dim length away until its shape was lost in the shadows. He wanted to explore its borders, but concluded that it would be best to sit down and rest awhile, first. Now, for the first time, the deep stillness of the place laid a clammy hand upon the spirits of the children. Becky said: "Why, I didn't notice, but it seems ever so long since I heard any of the others."

"Come to think, Becky, we are away down below them -- and I don't know how far away north, or south, or east, or whichever it is. We couldn't hear them here."

Becky grew apprehensive.

"I wonder how long we've been down here, Tom? We better start back."

"Yes, I reckon we better. P'raps we better."

"Can you find the way, Tom? It's all a mixed-up crookedness to me."

"I reckon I could find it -- but then the bats. If they put our candles out it will be an awful fix. Let's try some other way, so as not to go through there."

"Well. But I hope we won't get lost. It would be so awful!" and the girl shuddered at the thought of the dreadful possibilities.

They started through a corridor, and traversed it in silence a long way, glancing at each new opening, to see if there was anything familiar about the look of it; but they were all strange. Every time Tom made an examination, Becky would watch his face for an encouraging sign, and he would say cheerily:

"Oh, it's all right. This ain't the one, but we'll come to it right away!"

But he felt less and less hopeful with each failure, and presently began to turn off into diverging avenues at sheer random, in desperate hope of finding the one that was wanted. He still said it was "all right," but there was such a leaden dread at his heart that the words had lost their ring and sounded just as if he had said, "All is lost!" Becky clung to his side in an anguish of fear, and tried hard to keep back the tears, but they would come. At last she said:

"Oh, Tom, never mind the bats, let's go back that way! We seem to get worse and worse off all the time."

"Listen!" said he.

Profound silence; silence so deep that even their breathings were conspicuous in the hush. Tom shouted. The call went echoing down the empty aisles and died out in the distance in a faint sound that resembled a ripple of mocking laughter.

"Oh, don't do it again, Tom, it is too horrid," said Becky.

"It is horrid, but I better, Becky; they might hear us, you know," and he shouted again.

The "might" was even a chillier horror than the ghostly laughter, it so confessed a perishing hope. The children stood still and listened; but there was no result. Tom turned upon the back track at once, and hurried his steps. It was but a little while before a certain indecision in his manner revealed another fearful fact to Becky -- he could not find his way back!

"Oh, Tom, you didn't make any marks!"

"Becky, I was such a fool! Such a fool! I never thought we might want to come back! No -- I can't find the way. It's all mixed up."

"Tom, Tom, we're lost! we're lost! We never can get out of this awful place! Oh, why did we ever leave the others!"

She sank to the ground and burst into such a frenzy of crying that Tom was appalled with the idea that she might die, or lose her reason. He sat down by her and put his arms around her; she buried her face in his bosom, she clung to him, she poured out her terrors, her unavailing regrets, and the far echoes turned them all to jeering laughter. Tom begged her to pluck up hope again, and she said she could not. He fell to blaming and abusing himself for getting her into this miserable situation; this had a better effect. She said she would try to hope again, she would get up and follow wherever he might lead if only he would not talk like that any more. For he was no more to blame than she, she said.

So they moved on again -- aimlessly -- simply at random -- all they could do was to move, keep moving. For a little while, hope made a show of reviving -- not with any reason to back it, but only because it is its nature to revive when the spring has not been taken out of it by age and familiarity with failure.

By-and-by Tom took Becky's candle and blew it out. This economy meant so much! Words were not needed. Becky understood, and her hope died again. She knew that Tom had a whole candle and three or four pieces in his pockets -- yet he must economize.

By-and-by, fatigue began to assert its claims; the children tried to pay attention, for it was dreadful to think of sitting down when time was grown to be so precious, moving, in some direction, in any direction, was at least progress and might bear fruit; but to sit down was to invite death and shorten its pursuit.

At last Becky's frail limbs refused to carry her farther. She sat down. Tom rested with her, and they talked of home, and the friends there, and the comfortable beds and, above all, the light! Becky cried, and Tom tried to think of some way of comforting her, but all his encouragements were grown threadbare with use, and sounded like sarcasms. Fatigue bore so heavily upon Becky that she drowsed off to sleep. Tom was grateful. He sat looking into her drawn face and saw it grow smooth and natural under the influence of pleasant dreams; and by-and-by a smile dawned and rested there. The peaceful face reflected somewhat of peace and healing into his own spirit, and his thoughts wandered away to bygone times and dreamy memories. While he was deep in his musings, Becky woke up with a breezy little laugh -- but it was stricken dead upon her lips, and a groan followed it.

"Oh, how could I sleep! I wish I never, never had waked! No! No, I don't, Tom! Don't look so! I won't say it again."

"I'm glad you've slept, Becky; you'll feel rested, now, and we'll find the way out."

"We can try, Tom; but I've seen such a beautiful country in my dream. I reckon we are going there."

"Maybe not, maybe not. Cheer up, Becky, and let's go on trying."

They rose up and wandered along, hand in hand and hopeless. They tried to estimate how long they had been in the cave, but all they knew was that it seemed days and weeks, and yet it was plain that this could not be, for their candles were not gone yet. A long time after this -- they could not tell how long -- Tom said they must go softly and listen for dripping water -- they must find a spring. They found one presently, and Tom said it was time to rest again. Both were cruelly tired, yet Becky said she thought she could go a little farther. She was surprised to hear Tom dissent. She could not understand it. They sat down, and Tom fastened his candle to the wall in front of them with some clay. Thought was soon busy; nothing was said for some time. Then Becky broke the silence:

"Tom, I am so hungry!"

Tom took something out of his pocket.

"Do you remember this?" said he.

Becky almost smiled.

"It's our wedding-cake, Tom."

"Yes -- I wish it was as big as a barrel, for it's all we've got."

"I saved it from the picnic for us to dream on, Tom, the way grown-up people do with wedding-cake -- but it'll be our --"

She dropped the sentence where it was. Tom divided the cake and Becky ate with good appetite, while Tom nibbled at his moiety. There was abundance of cold water to finish the feast with. By-and-by Becky suggested that they move on again. Tom was silent a moment. Then he said:

"Becky, can you bear it if I tell you something?"

Becky's face paled, but she thought she could.

"Well, then, Becky, we must stay here, where there's water to drink. That little piece is our last candle!"

Becky gave loose to tears and wailings. Tom did what he could to comfort her, but with little effect. At length Becky said:

"Tom!"

"Well, Becky?"

"They'll miss us and hunt for us!"

"Yes, they will! Certainly they will!"

"Maybe they're hunting for us now, Tom."

"Why, I reckon maybe they are. I hope they are."

"When would they miss us, Tom?"

"When they get back to the boat, I reckon."

"Tom, it might be dark then -- would they notice we hadn't come?"

"I don't know. But anyway, your mother would miss you as soon as they got home."

A frightened look in Becky's face brought Tom to his senses and he saw that he had made a blunder. Becky was not to have gone home that night! The children became silent and thoughtful. In a moment a new burst of grief from Becky showed Tom that the thing in his mind had struck hers also -- that the Sabbath morning might be half spent before Mrs. Thatcher discovered that Becky was not at Mrs. Harper's.

The children fastened their eyes upon their bit of candle and watched it melt slowly and pitilessly away; saw the half inch of wick stand alone at last; saw the feeble flame rise and fall, climb the thin column of smoke, linger at its top a

moment, and then -- the horror of utter darkness reigned!

How long afterward it was that Becky came to a slow consciousness that she was crying in Tom's arms, neither could tell. All that they knew was, that after what seemed a mighty stretch of time, both awoke out of a dead stupor of sleep and resumed their miseries once more. Tom said it might be Sunday, now -- maybe Monday. He tried to get Becky to talk, but her sorrows were too oppressive, all her hopes were gone. Tom said that they must have been missed long ago, and no doubt the search was going on. He would shout and maybe some one would come. He tried it; but in the darkness the distant echoes sounded so hideously that he tried it no more.

The hours wasted away, and hunger came to torment the captives again. A portion of Tom's half of the cake was left; they divided and ate it. But they seemed hungrier than before. The poor morsel of food only whetted desire.

By-and-by Tom said:

"Sh! Did you hear that?"

Both held their breath and listened. There was a sound like the faintest, far-off shout. Instantly Tom answered it, and leading Becky by the hand, started groping down the corridor in its direction. Presently he listened again; again the sound was heard, and apparently a little nearer.

"It's them!" said Tom; "they're coming! Come along, Becky -- we're all right now!"

The joy of the prisoners was almost overwhelming. Their speed was slow, however, because pitfalls were somewhat common, and had to be guarded against. They shortly came to one and had to stop. It might be three feet deep, it might be a hundred -- there was no passing it at any rate. Tom got down on his breast and reached as far down as he could. No bottom. They must stay there and wait until the searchers came. They listened; evidently the distant shoutings were growing more distant! a moment or two more and they had gone altogether. The heart-sinking misery of it! Tom whooped until he was hoarse, but it was of no use. He talked hopefully to Becky; but an age of anxious waiting passed and no sounds came again.

The children groped their way back to the spring. The weary time dragged on; they slept again, and awoke famished and woe-stricken. Tom believed it must be Tuesday by this time.

Now an idea struck him. There were some side passages near at hand. It would be better to explore some of these than bear the weight of the heavy time in idleness. He took a kite-line from his pocket, tied it to a projection, and he and Becky started, Tom in the lead, unwinding the line as he groped along. At the end of twenty steps the corridor ended in a "jumping-off place." Tom got down on his knees and felt below, and then as far around the corner as he could reach with his hands conveniently; he made an effort to stretch yet a little farther to the right, and at that moment, not twenty yards away, a human hand, holding a candle, appeared from behind a rock! Tom lifted up a glorious shout, and instantly that hand was followed by the body it belonged to -- Injun Joe's! Tom was paralyzed; he could not move. He was vastly gratified the next moment, to see the "Spaniard" take to his heels and get himself out of sight. Tom wondered that Joe had not recognized his voice and come over and killed him for testifying in court. But the echoes must have disguised the voice. Without doubt, that was it, he reasoned. Tom's fright weakened every muscle in his body. He said to himself that if he had strength enough to get back to the spring he would stay there, and nothing should tempt him to run the risk of meeting Injun Joe again. He was careful to keep from Becky what it was he had seen. He told her he had only shouted "for luck."

But hunger and wretchedness rise superior to fears in the long run. Another tedious wait at the spring and another long sleep brought changes. The children awoke tortured with a raging hunger. Tom believed that it must be Wednesday or Thursday or even Friday or Saturday, now, and that the search had been given over. He proposed to explore another passage. He felt willing to risk Injun Joe and all other terrors. But Becky was very weak. She had sunk into a dreary apathy and would not be roused. She said she would wait, now, where she was, and die -- it would not be long. She told Tom to go with the kite-line and explore if he chose; but she implored him to come back every little while and speak to her; and she made him promise that when the awful time came, he would stay by her and hold her hand until all was over.

Tom kissed her, with a choking sensation in his throat, and made a show of being confident of finding the searchers or an escape from the cave; then he took the kite-line in his hand and went groping down one of the passages on his hands and knees, distressed with hunger and sick with bodings of coming doom.

The Adventures of Tom Sawyer/Chapter XXXII

Tuesday afternoon came, and waned to the twilight. The village of St. Petersburg still mourned. The lost children had not been found. Public prayers had been offered up for them, and many and many a private prayer that had the petitioner's whole heart in it; but still no good news came from the cave. The majority of the searchers had given up the quest and gone back to their daily avocations, saying that it was plain the children could never be found. Mrs. Thatcher was very ill, and a great part of the time delirious. People said it was heartbreaking to hear her call her child, and raise her head and listen a whole minute at a time, then lay it wearily down again with a moan. Aunt Polly had drooped into a settled melancholy, and her gray hair had grown almost white. The village went to its rest on Tuesday night, sad and forlorn. Away in the middle of the night a wild peal burst from the village bells, and in a moment the streets were swarming with frantic half-clad people, who shouted, "Turn out! turn out! they're found! they're found!" Tin pans and horns were added to the din, the population massed itself and moved toward the river, met the children coming in an open carriage drawn by shouting citizens, thronged around it, joined its homeward march, and swept magnificently up the main street roaring huzzah after huzzah!

The village was illuminated; nobody went to bed again; it was the greatest night the little town had ever seen. During the first half-hour a procession of villagers filed through Judge Thatcher's house, seized the saved ones and kissed them, squeezed Mrs. Thatcher's hand, tried to speak but couldn't -- and drifted out raining tears all over the place.

Aunt Polly's happiness was complete, and Mrs. Thatcher's nearly so. It would be complete, however, as soon as the messenger dispatched with the great news to the cave should get the word to her husband. Tom lay upon a sofa with an eager auditory about him and told the history of the wonderful adventure, putting in many striking additions to adorn it withal; and closed with a description of how he left Becky and went on an exploring expedition; how he followed two avenues as far as his kite-line would reach; how he followed a third to the fullest stretch of the kite-line, and was about to turn back when he glimpsed a far-off speck that looked like daylight; dropped the line and groped toward it, pushed his head and shoulders through a small hole, and saw the broad Mississippi rolling by! And if it had only happened to be night he would not have seen that speck of daylight and would not have explored that passage any more! He told how he went back for Becky and broke the good news and she told him not to fret her with such stuff, for she was tired, and knew she was going to die, and wanted to. He described how he labored with her and convinced her; and how she almost died for joy when she had groped to where she actually saw the blue speck of daylight; how he pushed his way out at the hole and then helped her out; how they sat there and cried for gladness; how some men came along in a skiff and Tom hailed them and told them their situation and their famished condition; how the men didn't believe the wild tale at first, "because," said they, "you are five miles down the river below the valley the cave is in" -- then took them aboard, rowed to a house, gave them supper, made them rest till two or three hours after dark and then brought them home.

Before day-dawn, Judge Thatcher and the handful of searchers with him were tracked out, in the cave, by the twine clews they had strung behind them, and informed of the great news.

Three days and nights of toil and hunger in the cave were not to be shaken off at once, as Tom and Becky soon discovered. They were bedridden all of Wednesday and Thursday, and seemed to grow more and more tired and worn, all the time. Tom got about, a little, on Thursday, was down-town Friday, and nearly as whole as ever Saturday; but Becky did not leave her room until Sunday, and then she looked as if she had passed through a wasting illness.

Tom learned of Huck's sickness and went to see him on Friday, but could not be admitted to the bedroom; neither could he on Saturday or Sunday. He was admitted daily after that, but was warned to keep still about his adventure and introduce no exciting topic. The Widow Douglas stayed by to see that he obeyed. At home Tom learned of the Cardiff Hill event; also that the "ragged man's" body had eventually been found in the river near the ferry-landing; he had been drowned while trying to escape, perhaps.

About a fortnight after Tom's rescue from the cave, he started off to visit Huck, who had grown plenty strong enough, now, to hear exciting talk, and Tom had some that would interest him, he thought. Judge Thatcher's house was on Tom's way, and he stopped to see Becky. The Judge and some friends set Tom to talking, and some one asked him ironically if he wouldn't like to go to the cave again. Tom said he thought he wouldn't mind it. The Judge said:

"Well, there are others just like you, Tom, I've not the least doubt. But we have taken care of that. Nobody will get lost in that cave any more."

"Why?"

"Because I had its big door sheathed with boiler iron two weeks ago, and triple-locked -- and I've got the keys."

Tom turned as white as a sheet.

"What's the matter, boy! Here, run, somebody! Fetch a glass of water!"

The water was brought and thrown into Tom's face.

"Ah, now you're all right. What was the matter with you, Tom?"

"Oh, Judge, Injun Joe's in the cave!"

The Adventures of Tom Sawyer/Chapter XXXIII

Within a few minutes the news had spread, and a dozen skiff-loads of men were on their way to McDougal's cave, and the ferryboat, well filled with passengers, soon followed. Tom Sawyer was in the skiff that bore Judge Thatcher. When the cave door was unlocked, a sorrowful sight presented itself in the dim twilight of the place. Injun Joe lay stretched upon the ground, dead, with his face close to the crack of the door, as if his longing eyes had been fixed, to the latest moment, upon the light and the cheer of the free world outside. Tom was touched, for he knew by his own experience how this wretch had suffered. His pity was moved, but nevertheless he felt an abounding sense of relief and security, now, which revealed to him in a degree which he had not fully appreciated before how vast a weight of dread had been lying upon him since the day he lifted his voice against this bloody-minded outcast.

Injun Joe's bowie-knife lay close by, its blade broken in two. The great foundation-beam of the door had been chipped and hacked through, with tedious labor; useless labor, too, it was, for the native rock formed a sill outside it, and upon that stubborn material the knife had wrought no effect; the only damage done was to the knife itself. But if there had been no stony obstruction there the labor would have been useless still, for if the beam had been wholly cut away Injun Joe could not have squeezed his body under the door, and he knew it. So he had only hacked that place in order to be doing something -- in order to pass the weary time -- in order to employ his tortured faculties. Ordinarily one could find half a dozen bits of candle stuck around in the crevices of this vestibule, left there by tourists; but there were none now. The prisoner had searched them out and eaten them. He had also contrived to catch a few bats, and these, also, he had eaten, leaving only their claws. The poor unfortunate had starved to death. In one place, near at hand, a stalagmite had been slowly growing up from the ground for ages, builded by the water-drip from a stalactite overhead. The captive had broken off the stalagmite, and upon the stump had placed a stone, wherein he had scooped a shallow hollow to catch the precious drop that fell once in every three minutes with the dreary regularity of a clock-tick -- a dessertspoonful once in four and twenty hours. That drop was falling when the Pyramids were new; when Troy fell; when the foundations of Rome were laid when Christ was crucified; when the

Conqueror created the British empire; when Columbus sailed; when the massacre at Lexington was "news." It is falling now; it will still be falling when all these things shall have sunk down the afternoon of history, and the twilight of tradition, and been swallowed up in the thick night of oblivion. Has everything a purpose and a mission? Did this drop fall patiently during five thousand years to be ready for this flitting human insect's need? and has it another important object to accomplish ten thousand years to come? No matter. It is many and many a year since the hapless half-breed scooped out the stone to catch the priceless drops, but to this day the tourist stares longest at that pathetic stone and that slow-dropping water when he comes to see the wonders of McDougal's cave. Injun Joe's cup stands first in the list of the cavern's marvels; even "Aladdin's Palace" cannot rival it.

Injun Joe was buried near the mouth of the cave; and people flocked there in boats and wagons from the towns and from all the farms and hamlets for seven miles around; they brought their children, and all sorts of provisions, and confessed that they had had almost as satisfactory a time at the funeral as they could have had at the hanging.

This funeral stopped the further growth of one thing -- the petition to the governor for Injun Joe's pardon. The petition had been largely signed; many tearful and eloquent meetings had been held, and a committee of sappy women been appointed to go in deep mourning and wail around the governor, and implore him to be a merciful ass and trample his duty under foot. Injun Joe was believed to have killed five citizens of the village, but what of that? If he had been Satan himself there would have been plenty of weaklings ready to scribble their names to a pardon-petition, and drip a tear on it from their permanently impaired and leaky water-works.

The morning after the funeral Tom took Huck to a private place to have an important talk. Huck had learned all about Tom's adventure from the Welshman and the Widow Douglas, by this time, but Tom said he reckoned there was one thing they had not told him; that thing was what he wanted to talk about now. Huck's face saddened. He said:

"I know what it is. You got into No. 2 and never found anything but whiskey. Nobody told me it was you; but I just knowed it must 'a' ben you, soon as I heard 'bout that whiskey business; and I knowed you hadn't got the money becuz you'd 'a' got at me some way or other and told me even if you was mum to everybody else. Tom, something's always told me we'd never get holt of that swag."

"Why, Huck, I never told on that tavern-keeper. You know his tavern was all right the Saturday I went to the picnic. Don't you remember you was to watch there that night?"

"Oh yes! Why, it seems 'bout a year ago. It was that very night that I follered Injun Joe to the widder's."

"You followed him?"

"Yes -- but you keep mum. I reckon Injun Joe's left friends behind him, and I don't want 'em souring on me and doing me mean tricks. If it hadn't ben for me he'd be down in Texas now, all right."

Then Huck told his entire adventure in confidence to Tom, who had only heard of the Welshman's part of it before.

"Well," said Huck, presently, coming back to the main question, "whoever nipped the whiskey in No. 2, nipped the money, too, I reckon -- anyways it's a goner for us, Tom."

"Huck, that money wasn't ever in No. 2!"

"What!" Huck searched his comrade's face keenly. "Tom, have you got on the track of that money again?"

"Huck, it's in the cave!"

Huck's eyes blazed.

"Say it again, Tom."

"The money's in the cave!"

"Tom -- honest injun, now -- is it fun, or earnest?"

"Earnest, Huck -- just as earnest as ever I was in my life. Will you go in there with me and help get it out?"

"I bet I will! I will if it's where we can blaze our way to it and not get lost."

"Huck, we can do that without the least little bit of trouble in the world."

"Good as wheat! What makes you think the money's --"

"Huck, you just wait till we get in there. If we don't find it I'll agree to give you my drum and every thing I've got in the world. I will, by jings."

"All right -- it's a whiz. When do you say?"

"Right now, if you say it. Are you strong enough?"

"Is it far in the cave? I ben on my pins a little, three or four days, now, but I can't walk more'n a mile, Tom -- least I don't think I could."

"It's about five mile into there the way anybody but me would go, Huck, but there's a mighty short cut that they don't anybody but me know about. Huck, I'll take you right to it in a skiff. I'll float the skiff down there, and I'll pull it back again all by myself. You needn't ever turn your hand over."

"Less start right off, Tom."

"All right. We want some bread and meat, and our pipes, and a little bag or two, and two or three kite-strings, and some of these new-fangled things they call lucifer matches. I tell you, many's the time I wished I had some when I was in there before."

A trifle after noon the boys borrowed a small skiff from a citizen who was absent, and got under way at once. When they were several miles below "Cave Hollow," Tom said:

"Now you see this bluff here looks all alike all the way down from the cave hollow -- no houses, no wood-yards, bushes all alike. But do you see that white place up yonder where there's been a landslide? Well, that's one of my marks. We'll get ashore, now."

They landed.

"Now, Huck, where we're a-standing you could touch that hole I got out of with a fishing-pole. See if you can find it."

Huck searched all the place about, and found nothing. Tom proudly marched into a thick clump of sumach bushes and said:

"Here you are! Look at it, Huck; it's the snuggest hole in this country. You just keep mum about it. All along I've been wanting to be a robber, but I knew I'd got to have a thing like this, and where to run across it was the bother. We've got it now, and we'll keep it quiet, only we'll let Joe Harper and Ben Rogers in -- because of course there's got to be a Gang, or else there wouldn't be any style about it. Tom Sawyer's Gang -- it sounds splendid, don't it, Huck?"

"Well, it just does, Tom. And who'll we rob?"

"Oh, most anybody. Waylay people -- that's mostly the way."

"And kill them?"

"No, not always. Hive them in the cave till they raise a ransom."

"What's a ransom?"

"Money. You make them raise all they can, off'n their friends; and after you've kept them a year, if it ain't raised then you kill them. That's the general way. Only you don't kill the women. You shut up the women, but you don't kill them. They're always beautiful and rich, and awfully scared. You take their watches and things, but you always take your hat off and talk polite. They ain't anybody as polite as robbers -- you'll see that in any book. Well, the women get to loving you, and after they've been in the cave a week or two weeks they stop crying and after that you couldn't get them to leave. If you drove them out they'd turn right around and come back. It's so in all the books."

"Why, it's real bully, Tom. I believe it's better'n to be a pirate."

"Yes, it's better in some ways, because it's close to home and circuses and all that."

By this time everything was ready and the boys entered the hole, Tom in the lead. They toiled their way to the farther end of the tunnel, then made their spliced kite-strings fast and moved on. A few steps brought them to the spring, and Tom felt a shudder quiver all through him. He showed Huck the fragment of candle-wick perched on a lump of clay against the wall, and described how he and Becky had watched the flame struggle and expire.

The boys began to quiet down to whispers, now, for the stillness and gloom of the place oppressed their spirits. They went on, and presently entered and followed Tom's other corridor until they reached the "jumping-off place." The candles revealed the fact that it was not really a precipice, but only a steep clay hill twenty or thirty feet high. Tom whispered:

"Now I'll show you something, Huck."

He held his candle aloft and said:

"Look as far around the corner as you can. Do you see that? There -- on the big rock over yonder -- done with candle-smoke."

"Tom, it's a cross!"

"Now where's your Number Two? 'Under the cross,' hey? Right yonder's where I saw Injun Joe poke up his candle, Huck!"

Huck stared at the mystic sign awhile, and then said with a shaky voice:

"Tom, less git out of here!"

"What! and leave the treasure?"

"Yes -- leave it. Injun Joe's ghost is round about there, certain."

"No it ain't, Huck, no it ain't. It would ha'n't the place where he died -- away out at the mouth of the cave -- five mile from here."

"No, Tom, it wouldn't. It would hang round the money. I know the ways of ghosts, and so do you."

Tom began to fear that Huck was right. Misgivings gathered in his mind. But presently an idea occurred to him --

"Lookyhere, Huck, what fools we're making of ourselves! Injun Joe's ghost ain't a going to come around where there's a cross!"

The point was well taken. It had its effect.

"Tom, I didn't think of that. But that's so. It's luck for us, that cross is. I reckon we'll climb down there and have a hunt for that box."

Tom went first, cutting rude steps in the clay hill as he descended. Huck followed. Four avenues opened out of the small cavern which the great rock stood in. The boys examined three of them with no result. They found a small recess in the one nearest the base of the rock, with a pallet of blankets spread down in it; also an old suspender, some bacon rind, and the well-gnawed bones of two or three fowls. But there was no money-box. The lads searched and researched this place, but in vain. Tom said:

"He said under the cross. Well, this comes nearest to being under the cross. It can't be under the rock itself, because that sets solid on the ground."

They searched everywhere once more, and then sat down discouraged. Huck could suggest nothing. By-and-by Tom said:

"Lookyhere, Huck, there's footprints and some candle-grease on the clay about one side of this rock, but not on the other sides. Now, what's that for? I bet you the money is under the rock. I'm going to dig in the clay."

"That ain't no bad notion, Tom!" said Huck with animation.

Tom's "real Barlow" was out at once, and he had not dug four inches before he struck wood.

"Hey, Huck! -- you hear that?"

Huck began to dig and scratch now. Some boards were soon uncovered and removed. They had concealed a natural chasm which led under the rock. Tom got into this and held his candle as far under the rock as he could, but said he could not see to the end of the rift. He proposed to explore. He stooped and passed under; the narrow way descended gradually. He followed its winding course, first to the right, then to the left, Huck at his heels. Tom turned a short curve, by-and-by, and exclaimed:

"My goodness, Huck, lookyhere!"

It was the treasure-box, sure enough, occupying a snug little cavern, along with an empty powder-keg, a couple of guns in leather cases, two or three pairs of old moccasins, a leather belt, and some other rubbish well soaked with the water-drip.

"Got it at last!" said Huck, ploughing among the tarnished coins with his hand. "My, but we're rich, Tom!"

"Huck, I always reckoned we'd get it. It's just too good to believe, but we have got it, sure! Say -- let's not fool around here. Let's snake it out. Lemme see if I can lift the box."

It weighed about fifty pounds. Tom could lift it, after an awkward fashion, but could not carry it conveniently.

"I thought so," he said; "They carried it like it was heavy, that day at the ha'nted house. I noticed that. I reckon I was right to think of fetching the little bags along."

The money was soon in the bags and the boys took it up to the cross rock.

"Now less fetch the guns and things," said Huck.

"No, Huck -- leave them there. They're just the tricks to have when we go to robbing. We'll keep them there all the time, and we'll hold our orgies there, too. It's an awful snug place for orgies."

"What orgies?"

"I dono. But robbers always have orgies, and of course we've got to have them, too. Come along, Huck, we've been in here a long time. It's getting late, I reckon. I'm hungry, too. We'll eat and smoke when we get to the skiff."

They presently emerged into the clump of sumach bushes, looked warily out, found the coast clear, and were soon lunching and smoking in the skiff. As the sun dipped toward the horizon they pushed out and got under way. Tom skimmed up the shore through the long twilight, chatting cheerily with Huck, and landed shortly after dark.

"Now, Huck," said Tom, "we'll hide the money in the loft of the widow's woodshed, and I'll come up in the morning and we'll count it and divide, and then we'll hunt up a place out in the woods for it where it will be safe. Just you lay quiet here and watch the stuff till I run and hook Benny Taylor's little wagon; I won't be gone a minute."

He disappeared, and presently returned with the wagon, put the two small sacks into it, threw some old rags on top of them, and started off, dragging his cargo behind him. When the boys reached the Welshman's house, they stopped to rest. Just as they were about to move on, the Welshman stepped out and said:

"Hallo, who's that?"

"Huck and Tom Sawyer."

"Good! Come along with me, boys, you are keeping everybody waiting. Here -- hurry up, trot ahead -- I'll haul the wagon for you. Why, it's not as light as it might be. Got bricks in it? -- or old metal?"

"Old metal," said Tom.

"I judged so; the boys in this town will take more trouble and fool away more time hunting up six bits' worth of old iron to sell to the foundry than they would to make twice the money at regular work. But that's human nature -- hurry along, hurry along!"

The boys wanted to know what the hurry was about.

"Never mind; you'll see, when we get to the Widow Douglas'."

Huck said with some apprehension -- for he was long used to being falsely accused:

"Mr. Jones, we haven't been doing nothing."

The Welshman laughed.

"Well, I don't know, Huck, my boy. I don't know about that. Ain't you and the widow good friends?"

"Yes. Well, she's ben good friends to me, anyway."

"All right, then. What do you want to be afraid for?"

This question was not entirely answered in Huck's slow mind before he found himself pushed, along with Tom, into Mrs. Douglas' drawing-room. Mr. Jones left the wagon near the door and followed.

The place was grandly lighted, and everybody that was of any consequence in the village was there. The Thatchers were there, the Harpers, the Rogerses, Aunt Polly, Sid, Mary, the minister, the editor, and a great many more, and all dressed in their best. The widow received the boys as heartily as any one could well receive two such looking beings. They were covered with clay and candle-grease. Aunt Polly blushed crimson with humiliation, and frowned and shook her head at Tom. Nobody suffered half as much as the two boys did, however. Mr. Jones said:

"Tom wasn't at home, yet, so I gave him up; but I stumbled on him and Huck right at my door, and so I just brought them along in a hurry."

"And you did just right," said the widow. "Come with me, boys."

She took them to a bedchamber and said:

"Now wash and dress yourselves. Here are two new suits of clothes -- shirts, socks, everything complete. They're Huck's -- no, no thanks, Huck -- Mr. Jones bought one and I the other. But they'll fit both of you. Get into them. We'll wait -- come down when you are slicked up enough."

Then she left.

The Adventures of Tom Sawyer/Chapter XXXIV

Huck said: "Tom, we can slope, if we can find a rope. The window ain't high from the ground." "Shucks! what do you want to slope for?"

"Well, I ain't used to that kind of a crowd. I can't stand it. I ain't going down there, Tom."

"Oh, bother! It ain't anything. I don't mind it a bit. I'll take care of you."

Sid appeared.

"Tom," said he, "auntie has been waiting for you all the afternoon. Mary got your Sunday clothes ready, and everybody's been fretting about you. Say -- ain't this grease and clay, on your clothes?"

"Now, Mr. Sid, you jist 'tend to your own business. What's all this blow-out about, anyway?"

"It's one of the widow's parties that she's always having. This time it's for the Welshman and his sons, on account of that scrape they helped her out of the other night. And say -- I can tell you something, if you want to know."

"Well, what?"

"Why, old Mr. Jones is going to try to spring something on the people here to-night, but I overheard him tell auntie to-day about it, as a secret, but I reckon it's not much of a secret now. Everybody knows -- the widow, too, for all she tries to let on she don't. Mr. Jones was bound Huck should be here -- couldn't get along with his grand secret without Huck, you know!"

"Secret about what, Sid?"

"About Huck tracking the robbers to the widow's. I reckon Mr. Jones was going to make a grand time over his surprise, but I bet you it will drop pretty flat."

Sid chuckled in a very contented and satisfied way.

"Sid, was it you that told?"

"Oh, never mind who it was. somebody told -- that's enough."

"Sid, there's only one person in this town mean enough to do that, and that's you. If you had been in Huck's place you'd 'a' sneaked down the hill and never told anybody on the robbers. You can't do any but mean things, and you can't bear to see anybody praised for doing good ones. There -- no thanks, as the widow says" -- and Tom cuffed Sid's ears and helped him to the door with several kicks. "Now go and tell auntie if you dare -- and to-morrow you'll catch it!"

Some minutes later the widow's guests were at the supper-table, and a dozen children were propped up at little side-tables in the same room, after the fashion of that country and that day. At the proper time Mr. Jones made his little speech, in which he thanked the widow for the honor she was doing himself and his sons, but said that there was another person whose modesty --

And so forth and so on. He sprung his secret about Huck's share in the adventure in the finest dramatic manner he was master of, but the surprise it occasioned was largely counterfeit and not as clamorous and effusive as it might have been under happier circumstances. However, the widow made a pretty fair show of astonishment, and heaped so many compliments and so much gratitude upon Huck that he almost forgot the nearly intolerable discomfort of his new clothes in the entirely intolerable discomfort of being set up as a target for everybody's gaze and everybody's laudations.

The widow said she meant to give Huck a home under her roof and have him educated; and that when she could spare the money she would start him in business in a modest way. Tom's chance was come. He said:

"Huck don't need it. Huck's rich."

Nothing but a heavy strain upon the good manners of the company kept back the due and proper complimentary laugh at this pleasant joke. But the silence was a little awkward. Tom broke it:

"Huck's got money. Maybe you don't believe it, but he's got lots of it. Oh, you needn't smile -- I reckon I can show you. You just wait a minute."

Tom ran out of doors. The company looked at each other with a perplexed interest -- and inquiringly at Huck, who was tongue-tied.

"Sid, what ails Tom?" said Aunt Polly. "He -- well, there ain't ever any making of that boy out. I never --"

Tom entered, struggling with the weight of his sacks, and Aunt Polly did not finish her sentence. Tom poured the mass of yellow coin upon the table and said:

"There -- what did I tell you? Half of it's Huck's and half of it's mine!"

The spectacle took the general breath away. All gazed, nobody spoke for a moment. Then there was a unanimous call for an explanation. Tom said he could furnish it, and he did. The tale was long, but brimful of interest. There was scarcely an interruption from any one to break the charm of its flow. When he had finished, Mr. Jones said:

"I thought I had fixed up a little surprise for this occasion, but it don't amount to anything now. This one makes it sing mighty small, I'm willing to allow."

The money was counted. The sum amounted to a little over twelve thousand dollars. It was more than any one present had ever seen at one time before, though several persons were there who were worth considerably more than that in property.

The Adventures of Tom Sawyer/Chapter XXXV

The reader may rest satisfied that Tom's and Huck's windfall made a mighty stir in the poor little village of St. Petersburg. So vast a sum, all in actual cash, seemed next to incredible. It was talked about, gloated over, glorified, until the reason of many of the citizens tottered under the strain of the unhealthy excitement. Every "haunted" house in St. Petersburg and the neighboring villages was dissected, plank by plank, and its foundations dug up and ransacked for hidden treasure -- and not by boys, but men -- pretty grave, unromantic men, too, some of them. Wherever Tom and Huck appeared they were courted, admired, stared at. The boys were not able to remember that their remarks had possessed weight before; but now their sayings were treasured and repeated; everything they did seemed somehow to be regarded as remarkable; they had evidently lost the power of doing and saying commonplace things; moreover, their past history was raked up and discovered to bear marks of conspicuous originality. The village paper published biographical sketches of the boys. The Widow Douglas put Huck's money out at six per cent., and Judge Thatcher did the same with Tom's at Aunt Polly's request. Each lad had an income, now, that was simply prodigious -- a dollar for every week-day in the year and half of the Sundays. It was just what the minister got -- no, it was what he was promised -- he generally couldn't collect it. A dollar and a quarter a week would board, lodge, and school a boy in those old simple days -- and clothe him and wash him, too, for that matter.

Judge Thatcher had conceived a great opinion of Tom. He said that no commonplace boy would ever have got his daughter out of the cave. When Becky told her father, in strict confidence, how Tom had taken her whipping at school, the Judge was visibly moved; and when she pleaded grace for the mighty lie which Tom had told in order to shift that whipping from her shoulders to his own, the Judge said with a fine outburst that it was a noble, a generous, a magnanimous lie -- a lie that was worthy to hold up its head and march down through history breast to breast with George Washington's lauded Truth about the hatchet! Becky thought her father had never looked so tall and so superb as when he walked the floor and stamped his foot and said that. She went straight off and told Tom about it.

Judge Thatcher hoped to see Tom a great lawyer or a great soldier some day. He said he meant to look to it that Tom should be admitted to the National Military Academy and afterward trained in the best law school in the country, in order that he might be ready for either career or both.

Huck Finn's wealth and the fact that he was now under the Widow Douglas' protection introduced him into society -- no, dragged him into it, hurled him into it -- and his sufferings were almost more than he could bear. The widow's servants kept him clean and neat, combed and brushed, and they bedded him nightly in unsympathetic sheets that had not one little spot or stain which he could press to his heart and know for a friend. He had to eat with a knife and fork; he had to use napkin, cup, and plate; he had to learn his book, he had to go to church; he had to talk so properly that speech was become insipid in his mouth; whithersoever he turned, the bars and shackles of civilization shut him in and bound him hand and foot.

He bravely bore his miseries three weeks, and then one day turned up missing. For forty-eight hours the widow hunted for him everywhere in great distress. The public were profoundly concerned; they searched high and low, they dragged the river for his body. Early the third morning Tom Sawyer wisely went poking among some old empty hogsheads down behind the abandoned slaughter-house, and in one of them he found the refugee. Huck had slept there; he had just breakfasted upon some stolen odds and ends of food, and was lying off, now, in comfort, with his pipe. He was unkempt, uncombed, and clad in the same old ruin of rags that had made him picturesque in the days when he was free and happy. Tom routed him out, told him the trouble he had been causing, and urged him to go home. Huck's face lost its tranquil content, and took a melancholy cast. He said:

"Don't talk about it, Tom. I've tried it, and it don't work; it don't work, Tom. It ain't for me; I ain't used to it. The widder's good to me, and friendly; but I can't stand them ways. She makes me get up just at the same time every morning; she makes me wash, they comb me all to thunder; she won't let me sleep in the woodshed; I got to wear them blamed clothes that just smothers me, Tom; they don't seem to any air git through 'em, somehow; and they're so rotten nice that I can't set down, nor lay down, nor roll around anywhar's; I hain't slid on a cellar-door for -- well, it

'pears to be years; I got to go to church and sweat and sweat -- I hate them ornery sermons! I can't ketch a fly in there, I can't chaw. I got to wear shoes all Sunday. The widder eats by a bell; she goes to bed by a bell; she gits up by a bell -- everything's so awful reg'lar a body can't stand it."

"Well, everybody does that way, Huck."

"Tom, it don't make no difference. I ain't everybody, and I can't stand it. It's awful to be tied up so. And grub comes too easy -- I don't take no interest in vittles, that way. I got to ask to go a-fishing; I got to ask to go in a-swimming -- dern'd if I hain't got to ask to do everything. Well, I'd got to talk so nice it wasn't no comfort -- I'd got to go up in the attic and rip out awhile, every day, to git a taste in my mouth, or I'd a died, Tom. The widder wouldn't let me smoke; she wouldn't let me yell, she wouldn't let me gape, nor stretch, nor scratch, before folks --" [Then with a spasm of special irritation and injury] -- "And dad fetch it, she prayed all the time! I never see such a woman! I had to shove, Tom -- I just had to. And besides, that school's going to open, and I'd a had to go to it -- well, I wouldn't stand that, Tom. Looky-here, Tom, being rich ain't what it's cracked up to be. It's just worry and worry, and sweat and sweat, and a-wishing you was dead all the time. Now these clothes suits me, and this bar'l suits me, and I ain't ever going to shake 'em any more. Tom, I wouldn't ever got into all this trouble if it hadn't 'a' ben for that money; now you just take my sheer of it along with your'n, and gimme a ten-center sometimes -- not many times, becuz I don't give a dern for a thing 'thout it's tollable hard to git -- and you go and beg off for me with the widder."

"Oh, Huck, you know I can't do that. 'Tain't fair; and besides if you'll try this thing just a while longer you'll come to like it."

"Like it! Yes -- the way I'd like a hot stove if I was to set on it long enough. No, Tom, I won't be rich, and I won't live in them cussed smothery houses. I like the woods, and the river, and hogsheads, and I'll stick to 'em, too. Blame it all! just as we'd got guns, and a cave, and all just fixed to rob, here this dern foolishness has got to come up and spile it all!"

Tom saw his opportunity --

"Lookyhere, Huck, being rich ain't going to keep me back from turning robber."

"No! Oh, good-licks; are you in real dead-wood earnest, Tom?"

"Just as dead earnest as I'm sitting here. But Huck, we can't let you into the gang if you ain't respectable, you know."

Huck's joy was quenched.

"Can't let me in, Tom? Didn't you let me go for a pirate?"

"Yes, but that's different. A robber is more high-toned than what a pirate is -- as a general thing. In most countries they're awful high up in the nobility -- dukes and such."

"Now, Tom, hain't you always ben friendly to me? You wouldn't shet me out, would you, Tom? You wouldn't do that, now, would you, Tom?"

"Huck, I wouldn't want to, and I don't want to -- but what would people say? Why, they'd say, 'Mph! Tom Sawyer's Gang! pretty low characters in it!' They'd mean you, Huck. You wouldn't like that, and I wouldn't."

Huck was silent for some time, engaged in a mental struggle. Finally he said:

"Well, I'll go back to the widder for a month and tackle it and see if I can come to stand it, if you'll let me b'long to the gang, Tom."

"All right, Huck, it's a whiz! Come along, old chap, and I'll ask the widow to let up on you a little, Huck."

"Will you, Tom -- now will you? That's good. If she'll let up on some of the roughest things, I'll smoke private and cuss private, and crowd through or bust. When you going to start the gang and turn robbers?"

"Oh, right off. We'll get the boys together and have the initiation to-night, maybe."

"Have the which?"

"Have the initiation."

"What's that?"

"It's to swear to stand by one another, and never tell the gang's secrets, even if you're chopped all to flinders, and kill anybody and all his family that hurts one of the gang."

"That's gay -- that's mighty gay, Tom, I tell you."

"Well, I bet it is. And all that swearing's got to be done at midnight, in the lonesomest, awfulest place you can find -- a ha'nted house is the best, but they're all ripped up now."

"Well, midnight's good, anyway, Tom."

"Yes, so it is. And you've got to swear on a coffin, and sign it with blood."

"Now, that's something like! Why, it's a million times bullier than pirating. I'll stick to the widder till I rot, Tom; and if I git to be a reg'lar ripper of a robber, and everybody talking 'bout it, I reckon she'll be proud she snaked me in out of the wet."

The Adventures of Tom Sawyer/Conclusion

So endeth this chronicle. It being strictly a history of a boy, it must stop here; the story could not go much further without becoming the history of a man. When one writes a novel about grown people, he knows exactly where to stop -- that is, with a marriage; but when he writes of juveniles, he must stop where he best can.

Most of the characters that perform in this book still live, and are prosperous and happy. Some day it may seem worth while to take up the story of the younger ones again and see what sort of men and women they turned out to be; therefore it will be wisest not to reveal any of that part of their lives at present.

Article Sources and Contributors

The Adventures of Tom Sawyer *Source:* <http://en.wikisource.org/w/index.php?oldid=3726766> *Contributors:* AdamBMorgan, Beeswaxcandle, Cowardly Lion, John Vandenberg, Jusjih, Kempm, Mikepjones, Museo8bits, Oleandr, Pmsyyz, Politicaljunkie, Spangineer, TayyabSaeed, Theornamentalist, Zhaladshar, 21 anonymous edits

The Adventures of Tom Sawyer/Chapter I *Source:* <http://en.wikisource.org/w/index.php?oldid=3727757> *Contributors:* BirgitteSB, DaL33T, George Orwell III, Oleandr, Zhaladshar, 5 anonymous edits

The Adventures of Tom Sawyer/Chapter II *Source:* <http://en.wikisource.org/w/index.php?oldid=3727756> *Contributors:* DaL33T, George Orwell III, Oleandr, Prosfilaes, Zhaladshar, 3 anonymous edits

The Adventures of Tom Sawyer/Chapter III *Source:* <http://en.wikisource.org/w/index.php?oldid=3727758> *Contributors:* DaL33T, George Orwell III, Jhanikhilesh87, Oleandr, Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter IV *Source:* <http://en.wikisource.org/w/index.php?oldid=3727759> *Contributors:* DaL33T, George Orwell III, Oleandr, Zhaladshar, 2 anonymous edits

The Adventures of Tom Sawyer/Chapter V *Source:* <http://en.wikisource.org/w/index.php?oldid=3748518> *Contributors:* DaL33T, Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter VI *Source:* <http://en.wikisource.org/w/index.php?oldid=3748519> *Contributors:* DaL33T, SCEhardt, Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter VII *Source:* <http://en.wikisource.org/w/index.php?oldid=3748520> *Contributors:* DaL33T, Zhaladshar, 2 anonymous edits

The Adventures of Tom Sawyer/Chapter VIII *Source:* <http://en.wikisource.org/w/index.php?oldid=3748521> *Contributors:* DaL33T, Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter IX *Source:* <http://en.wikisource.org/w/index.php?oldid=3748522> *Contributors:* DaL33T, Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter X *Source:* <http://en.wikisource.org/w/index.php?oldid=3748523> *Contributors:* DaL33T, Zhaladshar, 2 anonymous edits

The Adventures of Tom Sawyer/Chapter XI *Source:* <http://en.wikisource.org/w/index.php?oldid=3748524> *Contributors:* DaL33T, Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter XII *Source:* <http://en.wikisource.org/w/index.php?oldid=3748525> *Contributors:* DaL33T, Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter XIII *Source:* <http://en.wikisource.org/w/index.php?oldid=3748526> *Contributors:* Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter XIV *Source:* <http://en.wikisource.org/w/index.php?oldid=3748527> *Contributors:* Zhaladshar, 2 anonymous edits

The Adventures of Tom Sawyer/Chapter XV *Source:* <http://en.wikisource.org/w/index.php?oldid=3748528> *Contributors:* Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter XVI *Source:* <http://en.wikisource.org/w/index.php?oldid=3748529> *Contributors:* Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter XVII *Source:* <http://en.wikisource.org/w/index.php?oldid=3748530> *Contributors:* Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter XVIII *Source:* <http://en.wikisource.org/w/index.php?oldid=3748531> *Contributors:* Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter XIX *Source:* <http://en.wikisource.org/w/index.php?oldid=3748532> *Contributors:* Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter XX *Source:* <http://en.wikisource.org/w/index.php?oldid=3748533> *Contributors:* Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter XXI *Source:* <http://en.wikisource.org/w/index.php?oldid=3748534> *Contributors:* Politicaljunkie, Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter XXII *Source:* <http://en.wikisource.org/w/index.php?oldid=3748535> *Contributors:* Politicaljunkie, Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter XXIII *Source:* <http://en.wikisource.org/w/index.php?oldid=3748536> *Contributors:* Politicaljunkie, Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter XXIV *Source:* <http://en.wikisource.org/w/index.php?oldid=3748537> *Contributors:* Politicaljunkie, Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter XXV *Source:* <http://en.wikisource.org/w/index.php?oldid=3748538> *Contributors:* Politicaljunkie, Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter XXVI *Source:* <http://en.wikisource.org/w/index.php?oldid=3748539> *Contributors:* Billinghurst, Politicaljunkie, Remember the dot, Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter XXVII *Source:* <http://en.wikisource.org/w/index.php?oldid=3748540> *Contributors:* Politicaljunkie, Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter XXVIII *Source:* <http://en.wikisource.org/w/index.php?oldid=3748541> *Contributors:* Politicaljunkie, Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter XXIX *Source:* <http://en.wikisource.org/w/index.php?oldid=3748542> *Contributors:* Billinghurst, Politicaljunkie, Remember the dot, Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter XXX *Source:* <http://en.wikisource.org/w/index.php?oldid=3748543> *Contributors:* Billinghurst, Politicaljunkie, Remember the dot, Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter XXXI *Source:* <http://en.wikisource.org/w/index.php?oldid=3748544> *Contributors:* Politicaljunkie, Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter XXXII *Source:* <http://en.wikisource.org/w/index.php?oldid=3748545> *Contributors:* Politicaljunkie, Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter XXXIII *Source:* <http://en.wikisource.org/w/index.php?oldid=3748546> *Contributors:* Politicaljunkie, Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter XXXIV *Source:* <http://en.wikisource.org/w/index.php?oldid=3748547> *Contributors:* Politicaljunkie, Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Chapter XXXV *Source:* <http://en.wikisource.org/w/index.php?oldid=3748548> *Contributors:* Politicaljunkie, Zhaladshar, 1 anonymous edits

The Adventures of Tom Sawyer/Conclusion *Source:* <http://en.wikisource.org/w/index.php?oldid=3748549> *Contributors:* Ketamino, Politicaljunkie, 1 anonymous edits

Image Sources, Licenses and Contributors

Image:Speaker Icon.svg *Source:* http://en.wikisource.org/w/index.php?title=File:Speaker_Icon.svg *License:* Public Domain *Contributors:* Blast, G.Hagedorn, Mobius, Tehdog, 2 anonymous edits

File:PD-icon.svg *Source:* <http://en.wikisource.org/w/index.php?title=File:PD-icon.svg> *License:* Public Domain *Contributors:* Various. See log. (Original SVG was based on File:PD-icon.png by Duesentrieb, which was based on Image:Red copyright.png by Rfl.)

William Strunk, Jr.
The Elements of Style

NEW YORK 1918

Contents

PREFACE	III
I INTRODUCTORY	1
II ELEMENTARY RULES OF USAGE	3
1. Form the possessive singular of nouns with 's	3
2. In a series of three or more terms with a single conjunction, use a comma after each term except the last	4
3. Enclose parenthetic expressions between commas	4
4. Place a comma before <i>and</i> or <i>but</i> introducing an independent clause	6
5. Do not join independent clauses by a comma	7
6. Do not break sentences in two	8
7. A participial phrase at the beginning of a sentence must refer to the grammatical subject	9
8. Divide words at line-ends, in accordance with their formation and pronunciation .	10
III ELEMENTARY PRINCIPLES OF COMPOSITION	13
9. Make the paragraph the unit of composition: one paragraph to each topic	13
10. As a rule, begin each paragraph with a topic sentence; end it in conformity with the beginning	15
11. Use the active voice	18
12. Put statements in positive form	20
13. Omit needless words	21
14. Avoid a succession of loose sentences	23
15. Express co-ordinate ideas in similar form	24
16. Keep related words together	25
17. In summaries, keep to one tense	27
18. Place the emphatic words of a sentence at the end	28
IV A FEW MATTERS OF FORM	31
V WORDS AND EXPRESSIONS COMMONLY MISUSED	33
VI WORDS OFTEN MISSPELLED	43

PREFACE

Asserting that one must first know the rules to break them, this classic reference is a must-have for any student and conscientious writer. Intended for use in which the practice of composition is combined with the study of literature, it gives in brief space the principal requirements of plain English style and concentrates attention on the rules of usage and principles of composition most commonly violated.

Chapter I

INTRODUCTORY

This book is intended for use in English courses in which the practice of composition is combined with the study of literature. It aims to give in brief space the principal requirements of plain English style. It aims to lighten the task of instructor and student by concentrating attention (in Chapters II and III) on a few essentials, the rules of usage and principles of composition most commonly violated. The numbers of the sections may be used as references in correcting manuscript.

The book covers only a small portion of the field of English style, but the experience of its writer has been that once past the essentials, students profit most by individual instruction based on the problems of their own work, and that each instructor has his own body of theory, which he prefers to that offered by any textbook.

The writer's colleagues in the Department of English in Cornell University have greatly helped him in the preparation of his manuscript. Mr. George McLane Wood has kindly consented to the inclusion under Rule 11 of some material from his *Suggestions to Authors*.

The following books are recommended for reference or further study: in connection with Chapters II and IV, F. Howard Collins, *Author and Printer* (Henry Frowde); Chicago University Press, *Manual of Style*; T. L. De Vinne, *Correct Composition* (The Century Company); Horace Hart, *Rules for Compositors and Printers* (Oxford University Press); George McLane Wood, *Extracts from the Style-Book of the Government Printing Office* (United States Geological Survey); in connection with Chapters III and V, Sir Arthur Quiller-Couch, *The Art of Writing* (Putnams), especially the chapter, *Interlude on Jargon*; George McLane Wood, *Suggestions to Authors* (United States Geological Survey); John Leslie Hall, *English Usage* (Scott, Foresman and Co.); James P. Kelly, *Workmanship in Words* (Little, Brown and Co.).

It is an old observation that the best writers sometimes disregard the rules of rhetoric. When they do so, however, the reader will usually find in the sentence some compensating merit, attained at the cost of the violation. Unless he is certain of doing as well, he will probably do best to follow the rules. After he has learned, by their guidance, to write plain English adequate for everyday uses, let him look, for the secrets of style, to the study of the masters of literature.

Chapter II

ELEMENTARY RULES OF USAGE

1. Form the possessive singular of nouns with 's

Follow this rule whatever the final consonant. Thus write,

Charles's friend
Burns's poems
the witch's malice

This is the usage of the United States Government Printing Office and of the Oxford University Press.

Exceptions are the possessives of ancient proper names in -es and -is, the possessive Jesus', and such forms as *for conscience' sake*, *for righteousness' sake*. But such forms as *Achilles' heel*, *Moses' laws*, *Isis' temple* are commonly replaced by

the heel of Achilles
the laws of Moses
the temple of Isis

The pronominal possessives *hers*, *its*, *theirs*, *yours*, and *oneself* have no apostrophe.

2. In a series of three or more terms with a single conjunction, use a comma after each term except the last

Thus write,

red, white, and blue
honest, energetic, but headstrong
He opened the letter, read it and made a note of its contents.

This is also the usage of the Government Printing Office and of the Oxford University Press.

In the names of business firms the last comma is omitted, as

Brown, Shipley and Company

The abbreviation *etc.*, even if only a single term comes before it, is always preceded by a comma.

3. Enclose parenthetic expressions between commas

The best way to see a country, unless you are pressed for time, is to travel on foot.

This rule is difficult to apply; it is frequently hard to decide whether a single word, such as *however*, or a brief phrase, is or is not parenthetic. If the interruption to the flow of the sentence is but slight, the writer may safely omit the commas. But whether the interruption be slight or considerable, he must never omit one comma and leave the other. Such punctuation as

Marjorie's husband, Colonel Nelson paid us a visit yesterday.

My brother you will be pleased to hear, is now in perfect health.

is indefensible.

Non-restrictive relative clauses are, in accordance with this rule, set off by commas.

The audience, which had at first been indifferent, became more and more interested.

Similar clauses introduced by *where* and *when* are similarly punctuated.

In 1769, when Napoleon was born, Corsica had but recently been acquired by France.
--

Nether Stowey, where Coleridge wrote The Rime of the Ancient Mariner, is a few miles from Bridgewater.
--

In these sentences the clauses introduced by *which*, *when*, and *where* are non-restrictive; they do not limit the application of the words on which they depend, but add, parenthetically, statements supplementing those in the principal clauses. Each sentence is a combination of two statements which might have been made independently.

The audience was at first indifferent. Later it became more and more interested.
--

Napoleon was born in 1769. At that time Corsica had but recently been acquired by France.

Coleridge wrote The Rime of the Ancient Mariner at Nether Stowey. Nether Stowey is only a few miles from Bridgewater.

Restrictive relative clauses are not set off by commas.

The candidate who best meets these requirements will obtain the place.
--

In this sentence the relative clause restricts the application of the word *candidate* to a single person. Unlike those above, the sentence cannot be split into two independent statements.

The abbreviations *etc.* and *jr.* are always preceded by a comma, and except at the end of a sentence, followed by one.

Similar in principle to the enclosing of parenthetical expressions between commas is the setting off by commas of phrases or dependent clauses preceding or following the main clause of a sentence. The sentences quoted in this section and under Rules 4, 5, 6, 7, 16, and 18 should afford sufficient guidance.

If a parenthetical expression is preceded by a conjunction, place the first comma before the conjunction, not after it.

He saw us coming, and unaware that we had learned of his treachery, greeted us with a smile.
--

4. Place a comma before *and* or *but* introducing an independent clause

The early records of the city have disappeared, and the story of its first years can no longer be reconstructed.
--

The situation is perilous, but there is still one chance of escape.

Sentences of this type, isolated from their context, may seem to be in need of rewriting. As they make complete sense when the comma is reached, the second clause has the appearance of an after-thought. Further, *and*, is the least specific of connectives. Used between independent clauses, it indicates only that a relation exists between them without defining that relation. In the example above, the relation is that of cause and result. The two sentences might be rewritten:

As the early records of the city have disappeared, the story of its first years can no longer be reconstructed.

Although the situation is perilous, there is still one chance of escape.
--

Or the subordinate clauses might be replaced by phrases:

Owing to the disappearance of the early records of the city, the story of its first years can no longer be reconstructed.

In this perilous situation, there is still one chance of escape.
--

But a writer may err by making his sentences too uniformly compact and periodic, and an occasional loose sentence prevents the style from becoming too formal and gives the reader a certain relief. Consequently, loose sentences of the type first quoted are

common in easy, unstudied writing. But a writer should be careful not to construct too many of his sentences after this pattern (see Rule 14).

Two-part sentences of which the second member is introduced by *as* (in the sense of *because*), *for*, *or*, *nor*, and *while* (in the sense of *and at the same time*) likewise require a comma before the conjunction.

If a dependent clause, or an introductory phrase requiring to be set off by a comma, precedes the second independent clause, no comma is needed after the conjunction.

The situation is perilous, but if we are prepared to act promptly, there is still one chance of escape.

For two-part sentences connected by an adverb, see the next section.

5. Do not join independent clauses by a comma

If two or more clauses, grammatically complete and not joined by a conjunction, are to form a single compound sentence, the proper mark of punctuation is a semicolon.

Stevenson's romances are entertaining; they are full of exciting adventures.
--

It is nearly half past five; we cannot reach town before dark.
--

It is of course equally correct to write the above as two sentences each, replacing the semicolons by periods.

Stevenson's romances are entertaining. They are full of exciting adventures.
--

It is nearly half past five. We cannot reach town before dark.
--

If a conjunction is inserted, the proper mark is a comma (Rule 4).

Stevenson's romances are entertaining, for they are full of exciting adventures.
--

It is nearly half past five, and we cannot reach town before dark.
--

Note that if the second clause is preceded by an adverb, such as *accordingly*, *besides*, *so*, *then*, *therefore*, or *thus*, and not by a conjunction, the semicolon is still required.

I had never been in the place before; so I had difficulty in finding my way about.

In general, however, it is best, in writing, to avoid using *so* in this manner; there is danger that the writer who uses it at all may use it too often. A simple correction, usually serviceable, is to omit the word *so*, and begin the first clause with *as*:

As I had never been in the place before, I had difficulty in finding my way about.

If the clauses are very short, and are alike in form, a comma is usually permissible:

Man proposes, God disposes.

The gate swung apart, the bridge fell, the portcullis was drawn up.

6. Do not break sentences in two

In other words, do not use periods for commas.

I met them on a Cunard liner several years ago. Coming home from Liverpool to New York.

He was an interesting talker. A man who had traveled all over the world, and lived in half a dozen countries.

In both these examples, the first period should be replaced by a comma, and the following word begun with a small letter.

It is permissible to make an emphatic word or expression serve the purpose of a sentence and to punctuate it accordingly:

Again and again he called out. No reply.

The writer must, however, be certain that the emphasis is warranted, and that he will not be suspected of a mere blunder in punctuation.

Rules 3, 4, 5, and 6 cover the most important principles in the punctuation of ordinary sentences; they should be so thoroughly mastered that their application becomes second nature.

7. A participial phrase at the beginning of a sentence must refer to the grammatical subject

Walking slowly down the road, he saw a woman accompanied by two children.

The word *walking* refers to the subject of the sentence, not to the woman. If the writer wishes to make it refer to the woman, he must recast the sentence:

He saw a woman, accompanied by two children, walking slowly down the road.

Participial phrases preceded by a conjunction or by a preposition, nouns in apposition, adjectives, and adjective phrases come under the same rule if they begin the sentence.

On arriving in Chicago, his friends met him at the station.	When he arrived (or, On his arrival) in Chicago, his friends met him at the station.
A soldier of proved valor, they entrusted him with the defence of the city.	A soldier of proved valor, he was entrusted with the defence of the city.
Young and inexperienced, the task seemed easy to me.	Young and inexperienced, I thought the task easy.
Without a friend to counsel him, the temptation proved irresistible.	Without a friend to counsel him, he found the temptation irresistible.

Sentences violating this rule are often ludicrous.

Being in a dilapidated condition, I was able to buy the house very cheap.

8. Divide words at line-ends, in accordance with their formation and pronunciation

If there is room at the end of a line for one or more syllables of a word, but not for the whole word, divide the word, unless this involves cutting off only a single letter, or cutting off only two letters of a long word. No hard and fast rule for all words can be laid down. The principles most frequently applicable are:

A. Divide the word according to its formation:

know-ledge (not knowl-edge)

Shake-speare (not Shakes-peare)

de-scribe (not des-cribe)

atmo-sphere (not atmos-phere)

B. Divide "on the vowel:

edi-ble (not ed-ible)	propo-sition
ordi-nary	espe-cial
reli-gious	oppo-nents
regu-lar	classi-fi-ca-tion (three divisions possible)
deco-rative	presi-dent

C. Divide between double letters, unless they come at the end of the simple form of the word:

Apen-nines	Cincin-nati
refer-ring	tell-ing

The treatment of consonants in combination is best shown from examples:

for-tune	pic-ture
presump-tuous	illus-tration
sub-stan-tial (either division)	indus-try

instruc-tion	sug-ges-tion
incen-diary	

The student will do well to examine the syllable-division in a number of pages of any carefully printed book.

Chapter III

ELEMENTARY PRINCIPLES OF COMPOSITION

9. Make the paragraph the unit of composition: one paragraph to each topic

If the subject on which you are writing is of slight extent, or if you intend to treat it very briefly, there may be no need of subdividing it into topics. Thus a brief description, a brief summary of a literary work, a brief account of a single incident, a narrative merely outlining an action, the setting forth of a single idea, any one of these is best written in a single paragraph. After the paragraph has been written, it should be examined to see whether subdivision will not improve it.

Ordinarily, however, a subject requires subdivision into topics, each of which should be made the subject of a paragraph. The object of treating each topic in a paragraph by itself is, of course, to aid the reader. The beginning of each paragraph is a signal to him that a new step in the development of the subject has been reached.

The extent of subdivision will vary with the length of the composition. For example, a short notice of a book or poem might consist of a single paragraph. One slightly longer might consist of two paragraphs:

- A. Account of the work.
- B. Critical discussion.

14 CHAPTER III. ELEMENTARY PRINCIPLES OF COMPOSITION

A report on a poem, written for a class in literature, might consist of seven paragraphs:

- A. Facts of composition and publication.
- B. Kind of poem; metrical form.
- C. Subject.
- D. Treatment of subject.
- E. For what chiefly remarkable.
- F. Wherein characteristic of the writer.
- G. Relationship to other works.

The contents of paragraphs C and D would vary with the poem. Usually, paragraph C would indicate the actual or imagined circumstances of the poem (the situation), if these call for explanation, and would then state the subject and outline its development. If the poem is a narrative in the third person throughout, paragraph C need contain no more than a concise summary of the action. Paragraph D would indicate the leading ideas and show how they are made prominent, or would indicate what points in the narrative are chiefly emphasized.

A novel might be discussed under the heads:

- A. Setting.
- B. Plot.
- C. Characters.
- D. Purpose.

A historical event might be discussed under the heads:

- A. What led up to the event.
- B. Account of the event.
- C. What the event led up to.

In treating either of these last two subjects, the writer would probably find it necessary to subdivide one or more of the topics here given.

As a rule, single sentences should not be written or printed as paragraphs.

An exception may be made of sentences of transition, indicating the relation between the parts of an exposition or argument.

In dialogue, each speech, even if only a single word, is a paragraph by itself; that is, a new paragraph begins with each change of speaker. The application of this rule, when

dialogue and narrative are combined, is best learned from examples in well-printed works of fiction.

10. As a rule, begin each paragraph with a topic sentence; end it in conformity with the beginning

Again, the object is to aid the reader. The practice here recommended enables him to discover the purpose of each paragraph as he begins to read it, and to retain the purpose in mind as he ends it. For this reason, the most generally useful kind of paragraph, particularly in exposition and argument, is that in which

- A. the topic sentence comes at or near the beginning;
- B. the succeeding sentences explain or establish or develop the statement made in the topic sentence; and
- C. the final sentence either emphasizes the thought of the topic sentence or states some important consequence.

Ending with a digression, or with an unimportant detail, is particularly to be avoided.

If the paragraph forms part of a larger composition, its relation to what precedes, or its function as a part of the whole, may need to be expressed. This can sometimes be done by a mere word or phrase (*again; therefore; for the same reason*) in the topic sentence. Sometimes, however, it is expedient to precede the topic sentence by one or more sentences of introduction or transition. If more than one such sentence is required, it is generally better to set apart the transitional sentences as a separate paragraph.

According to the writer's purpose, he may, as indicated above, relate the body of the paragraph to the topic sentence in one or more of several different ways. He may make the meaning of the topic sentence clearer by restating it in other forms, by defining its terms, by denying the converse, by giving illustrations or specific instances; he may establish it by proofs; or he may develop it by showing its implications and consequences. In a long paragraph, he may carry out several of these processes.

1 Now, to be properly enjoyed, a walking tour should be gone upon alone.	1 Topic sentence.
2 If you go in a company, or even in pairs, it is no longer a walking tour in anything but name; it is something else and more in the nature of a picnic.	2 The meaning made clearer by denial of the contrary.

3 A walking tour should be gone upon alone, because freedom is of the essence; because you should be able to stop and go on, and follow this way or that, as the freak takes you; and because you must have your own pace, and neither trot alongside a champion walker, nor mince in time with a girl.	3 The topic sentence repeated, in abridged form, and supported by three reasons; the meaning of the third ("you must have your own pace") made clearer by denying the converse.
4 And you must be open to all impressions and let your thoughts take colour from what you see.	4 A fourth reason, stated in two forms.
5 You should be as a pipe for any wind to play upon.	5 The same reason, stated in still another form.
6 "I cannot see the wit," says Hazlitt, "of walking and talking at the same time." 7 When I am in the country, I wish to vegetate like the country, which is the gist of all that can be said upon the matter.	6-7 The same reason as stated by Hazlitt.
8 There should be no cackle of voices at your elbow, to jar on the meditative silence of the morning.	8 Repetition, in paraphrase, of the quotation from Hazlitt.
9 And so long as a man is reasoning he cannot surrender himself to that fine intoxication that comes of much motion in the open air, that begins in a sort of dazzle and sluggishness of the brain, and ends in a peace that passes comprehension. - Stevenson, Walking Tours.	9 Final statement of the fourth reason, in language amplified and heightened to form a strong conclusion.

1 It was chiefly in the eighteenth century that a very different conception of history grew up.	1 Topic sentence.
2 Historians then came to believe that their task was not so much to paint a picture as to solve a problem; to explain or illustrate the successive phases of national growth, prosperity, and adversity.	2 The meaning of the topic sentence made clearer; the new conception of history defined.
3 The history of morals, of industry, of intellect, and of art; the changes that take place in manners or beliefs; the dominant ideas that prevailed in successive periods; the rise, fall, and modification of political constitutions; in a word, all the conditions of national well-being became the subjects of their works.	3 The definition expanded.
4 They sought rather to write a history of peoples than a history of kings.	4 The definition explained by contrast.
5 They looked especially in history for the chain of causes and effects.	5 The definition supplemented: another element in the new conception of history.
6 They undertook to study in the past the physiology of nations, and hoped by applying the experimental method on a large scale to deduce some lessons of real value about the conditions on which the welfare of society mainly depend. -Lecky, The Political Value of History.	6 Conclusion: an important consequence of the new conception of history.

In narration and description the paragraph sometimes begins with a concise, comprehensive statement serving to hold together the details that follow.

The breeze served us admirably.
The campaign opened with a series of reverses.
The next ten or twelve pages were filled with a curious set of entries.

But this device, if too often used, would become a mannerism. More commonly the opening sentence simply indicates by its subject with what the paragraph is to be principally concerned.

At length I thought I might return towards the stockade.
He picked up the heavy lamp from the table and began to explore.
Another flight of steps, and they emerged on the roof.

The brief paragraphs of animated narrative, however, are often without even this semblance of a topic sentence. The break between them serves the purpose of a rhetorical pause, throwing into prominence some detail of the action.

11. Use the active voice

The active voice is usually more direct and vigorous than the passive:

I shall always remember my first visit to Boston.

This is much better than

My first visit to Boston will always be remembered by me.

The latter sentence is less direct, less bold, and less concise. If the writer tries to make it more concise by omitting "by me,"

My first visit to Boston will always be remembered,

it becomes indefinite: is it the writer, or some person undisclosed, or the world at large, that will always remember this visit?

This rule does not, of course, mean that the writer should entirely discard the passive

voice, which is frequently convenient and sometimes necessary.

The dramatists of the Restoration are little esteemed to-day.
Modern readers have little esteem for the dramatists of the Restoration.

The first would be the right form in a paragraph on the dramatists of the Restoration; the second, in a paragraph on the tastes of modern readers. The need of making a particular word the subject of the sentence will often, as in these examples, determine which voice is to be used.

The habitual use of the active voice, however, makes for forcible writing. This is true not only in narrative principally concerned with action, but in writing of any kind. Many a tame sentence of description or exposition can be made lively and emphatic by substituting a transitive in the active voice for some such perfunctory expression as there is, or could be heard.

There were a great number of dead leaves lying on the ground.	Dead leaves covered the ground.
The sound of the falls could still be heard.	The sound of the falls still reached our ears.
The reason that he left college was that his health became impaired.	Failing health compelled him to leave college.
It was not long before he was very sorry that he had said what he had.	He soon repented his words.

As a rule, avoid making one passive depend directly upon another.

Gold was not allowed to be exported.	It was forbidden to export gold (The export of gold was prohibited).
He has been proved to have been seen entering the building.	It has been proved that he was seen to enter the building.

In both the examples above, before correction, the word properly related to the second passive is made the subject of the first.

A common fault is to use as the subject of a passive construction a noun which ex-

20 CHAPTER III. ELEMENTARY PRINCIPLES OF COMPOSITION

presses the entire action, leaving to the verb no function beyond that of completing the sentence.

A survey of this region was made in 1900.	This region was surveyed in 1900.
Mobilization of the army was rapidly carried out.	The army was rapidly mobilized.
Confirmation of these reports cannot be obtained.	These reports cannot be confirmed.

Compare the sentence, "The export of gold was prohibited," in which the predicate "was prohibited" expresses something not implied in "export."

12. Put statements in positive form

Make definite assertions. Avoid tame, colorless, hesitating, non-committal language. Use the word *not* as a means of denial or in antithesis, never as a means of evasion.

He was not very often on time.	He usually came late.
He did not think that studying Latin was much use.	He thought the study of Latin useless.
<i>The Taming of the Shrew</i> is rather weak in spots. Shakespeare does not portray Katharine as a very admirable character, nor does Bianca remain long in memory as an important character in Shakespeare's works.	The women in <i>The Taming of the Shrew</i> are unattractive. Katharine is disagreeable, Bianca insignificant.

The last example, before correction, is indefinite as well as negative. The corrected version, consequently, is simply a guess at the writer's intention.

All three examples show the weakness inherent in the word *not*. Consciously or unconsciously, the reader is dissatisfied with being told only what is not; he wishes to be told what is. Hence, as a rule, it is better to express a negative in positive form.

not honest	dishonest
not important	trifling
did not remember	forgot
did not pay any attention to	ignored
did not have much confidence in	distrusted

The antithesis of negative and positive is strong:

Not charity, but simple justice.
Not that I loved Caesar less, but Rome the more.

Negative words other than *not* are usually strong:

The sun never sets upon the British flag.

13. Omit needless words

Vigorous writing is concise. A sentence should contain no unnecessary words, a paragraph no unnecessary sentences, for the same reason that a drawing should have no unnecessary lines and a machine no unnecessary parts. This requires not that the writer make all his sentences short, or that he avoid all detail and treat his subjects only in outline, but that every word tell.

Many expressions in common use violate this principle:

the question as to whether	whether (the question whether)
there is no doubt but that	no doubt (doubtless)
used for fuel purposes	used for fuel
he is a man who	he

22 CHAPTER III. ELEMENTARY PRINCIPLES OF COMPOSITION

in a hasty manner	hastily
this is a subject which	this subject
His story is a strange one.	His story is strange.

In especial the expression *the fact that* should be revised out of every sentence in which it occurs.

owing to the fact that	since (because)
in spite of the fact that	though (although)
call your attention to the fact that	remind you (notify you)
I was unaware of the fact that	I was unaware that (did not know)
the fact that he had not succeeded	his failure
the fact that I had arrived	my arrival

See also under *case*, *character*, *nature*, *system* in Chapter V.

Who is, *which was*, and the like are often superfluous.

His brother, who is a member of the same firm	His brother, a member of the same firm
Trafalgar, which was Nelson's last battle	Trafalgar, Nelson's last battle

As positive statement is more concise than negative, and the active voice more concise than the passive, many of the examples given under Rules 11 and 12 illustrate this rule as well.

A common violation of conciseness is the presentation of a single complex idea, step by step, in a series of sentences which might to advantage be combined into one.

Macbeth was very ambitious. This led him to wish to become king of Scotland. The witches told him that this wish of his would come true. The king of Scotland at this time was Duncan. Encouraged by his wife, Macbeth murdered Duncan. He was thus enabled to succeed Duncan as king. (55 words.)	Encouraged by his wife, Macbeth achieved his ambition and realized the prediction of the witches by murdering Duncan and becoming king of Scotland in his place. (26 words.)
--	--

14. Avoid a succession of loose sentences

This rule refers especially to loose sentences of a particular type, those consisting of two co-ordinate clauses, the second introduced by a conjunction or relative. Although single sentences of this type may be unexceptionable (see under Rule 4), a series soon becomes monotonous and tedious.

An unskilful writer will sometimes construct a whole paragraph of sentences of this kind, using as connectives *and*, *but*, and less frequently, *who*, *which*, *when*, *where*, and *while*, these last in non-restrictive senses (see under Rule 3).

The third concert of the subscription series was given last evening, and a large audience was in attendance. Mr. Edward Appleton was the soloist, and the Boston Symphony Orchestra furnished the instrumental music. The former showed himself to be an artist of the first rank, while the latter proved itself fully deserving of its high reputation. The interest aroused by the series has been very gratifying to the Committee, and it is planned to give a similar series annually hereafter. The fourth concert will be given on Tuesday, May 10, when an equally attractive programme will be presented.

Apart from its triteness and emptiness, the paragraph above is bad because of the structure of its sentences, with their mechanical symmetry and sing-song. Contrast with them the sentences in the paragraphs quoted under Rule 10, or in any piece of good English prose, as the preface (Before the Curtain) to *Vanity Fair*.

If the writer finds that he has written a series of sentences of the type described, he should recast enough of them to remove the monotony, replacing them by simple sentences, by sentences of two clauses joined by a semicolon, by periodic sentences of two clauses, by sentences, loose or periodic, of three clauses-whichever best represent the real relations of the thought. **15. Express co-ordinate ideas in similar form**

This principle, that of parallel construction, requires that expressions of similar content and function should be outwardly similar. The likeness of form enables the reader to recognize more readily the likeness of content and function. Familiar instances from the Bible are the Ten Commandments, the Beatitudes, and the petitions of the Lord's Prayer.

The unskilful writer often violates this principle, from a mistaken belief that he should constantly vary the form of his expressions. It is true that in repeating a statement in order to emphasize it he may have need to vary its form. For illustration, see the paragraph from Stevenson quoted under Rule 10. But apart from this, he should follow the principle of parallel construction.

Formerly, science was taught by the textbook method, while now the laboratory method is employed.	Formerly, science was taught by the textbook method; now it is taught by the laboratory method.
---	---

The left-hand version gives the impression that the writer is undecided or timid; he seems unable or afraid to choose one form of expression and hold to it. The right-hand version shows that the writer has at least made his choice and abided by it.

By this principle, an article or a preposition applying to all the members of a series must either be used only before the first term or else be repeated before each term.

The French, the Italians, Spanish, and Portuguese	The French, the Italians, the panish, and the Portuguese
In spring, summer, or in winter	In spring, summer, or winter (In spring, in summer, or in winter)

Correlative expressions (*both, and; not, but; not only, but also; either, or; first, second, third; and the like*) should be followed by the same grammatical construction. Many violations of this rule can be corrected by rearranging the sentence.

It was both a long ceremony and very tedious.	The ceremony was both long and tedious.
---	---

A time not for words, but action	A time not for words, but for action
Either you must grant his request or incur his ill will.	You must either grant his request or incur his ill will.
My objections are, first, the injustice of the measure; second, that it is unconstitutional.	My objections are, first, that the measure is unjust; second, that it is unconstitutional.

See also the third example under Rule 12 and the last under Rule 13.

It may be asked, what if a writer needs to express a very large number of similar ideas, say twenty? Must he write twenty consecutive sentences of the same pattern? On closer examination he will probably find that the difficulty is imaginary, that his twenty ideas can be classified in groups, and that he need apply the principle only within each group. Otherwise he had best avoid the difficulty by putting his statements in the form of a table.

16. Keep related words together

The position of the words in a sentence is the principal means of showing their relationship. The writer must therefore, so far as possible, bring together the words, and groups of words, that are related in thought, and keep apart those which are not so related.

The subject of a sentence and the principal verb should not, as a rule, be separated by a phrase or clause that can be transferred to the beginning.

Wordsworth, in the fifth book of <i>The Excursion</i> , gives a minute description of this church.	In the fifth book of <i>The Excursion</i> , Wordsworth gives a minute description of this church.
Cast iron, when treated in a Bessemer converter, is changed into steel.	By treatment in a Bessemer converter, cast iron is changed into steel.

The objection is that the interposed phrase or clause needlessly interrupts the natural order of the main clause. This objection, however, does not usually hold when the order is interrupted only by a relative clause or by an expression in apposition. Nor does it hold in periodic sentences in which the interruption is a deliberately used means of creating suspense (see examples under Rule 18).

The relative pronoun should come, as a rule, immediately after its antecedent.

There was a look in his eye that boded mischief.	In his eye was a look that boded mischief.
He wrote three articles about his adventures in Spain, which were published in <i>Harper's Magazine</i> .	He published in <i>Harper's Magazine</i> three articles about his adventures in Spain.
This is a portrait of Benjamin Harrison, grandson of William Henry Harrison, who became President in 1889.	This is a portrait of Benjamin Harrison, grandson of William Henry Harrison. He became President in 1889.

If the antecedent consists of a group of words, the relative comes at the end of the group, unless this would cause ambiguity.

The Superintendent of the Chicago Division, who	
A proposal to amend the Sherman Act, which has been variously judged	A proposal, which has been variously judged, to amend the Sherman Act <hr/> A proposal to amend the much-debated Sherman Act
The grandson of William Henry Harrison, who	William Henry Harrison's grandson, Benjamin Harrison, who

A noun in apposition may come between antecedent and relative, because in such a combination no real ambiguity can arise.

The Duke of York, his brother, who was regarded with hostility by the Whigs

Modifiers should come, if possible next to the word they modify. If several expressions modify the same word, they should be so arranged that no wrong relation is suggested.

All the members were not present.	Not all the members were present.
He only found two mistakes.	He found only two mistakes.

Major R. E. Joyce will give a lecture on Tuesday evening in Bailey Hall, to which the public is invited, on "My Experiences in Mesopotamia" at eight P.M.	On Tuesday evening at eight P.M., Major R.E. Joyce will give in Bailey Hall a lecture on "My Experiences in Mesopotamia." The public is invited.
---	--

17. In summaries, keep to one tense

In summarizing the action of a drama, the writer should always use the present tense. In summarizing a poem, story, or novel, he should preferably use the present, though he may use the past if he prefers. If the summary is in the present tense, antecedent action should be expressed by the perfect; if in the past, by the past perfect.

An unforeseen chance prevents Friar John from delivering Friar Lawrence's letter to Romeo. Juliet, meanwhile, owing to her father's arbitrary change of the day set for her wedding, has been compelled to drink the potion on Tuesday night, with the result that Balthasar informs Romeo of her supposed death before Friar Lawrence learns of the nondelivery of the letter.

But whichever tense be used in the summary, a past tense in indirect discourse or in indirect question remains unchanged.

The Legate inquires who struck the blow.

Apart from the exceptions noted, whichever tense the writer chooses, he should use throughout. Shifting from one tense to the other gives the appearance of uncertainty and irresolution (compare Rule 15).

In presenting the statements or the thought of some one else, as in summarizing an essay or reporting a speech, the writer should avoid intercalating such expressions as "he said," "he stated," "the speaker added," "the speaker then went on to say," "the author also thinks," or the like. He should indicate clearly at the outset, once for all, that what follows is summary, and then waste no words in repeating the notification. In notebooks, in newspapers, in handbooks of literature, summaries of one kind or another may be indispensable, and for children in primary schools it is a useful exercise to retell a story in their own words. But in the criticism or interpretation of literature the writer should be careful to avoid dropping into summary. He may find it necessary to devote one or two sentences to indicating the subject, or the opening situation, of the work he is discussing; he may cite numerous details to illustrate its qualities. But he should aim

to write an orderly discussion supported by evidence, not a summary with occasional comment. Similarly, if the scope of his discussion includes a number of works, he will as a rule do better not to take them up singly in chronological order, but to aim from the beginning at establishing general conclusions.

18. Place the emphatic words of a sentence at the end

The proper place for the word, or group of words, which the writer desires to make most prominent is usually the end of the sentence.

Humanity has hardly advanced in fortitude since that time, though it has advanced in many other ways.	Humanity, since that time, has advanced in many other ways, but it has hardly advanced in fortitude.
This steel is principally used for making razors, because of its hardness.	Because of its hardness, this steel is principally used in making razors.

The word or group of words entitled to this position of prominence is usually the logical predicate, that is, the new element in the sentence, as it is in the second example.

The effectiveness of the periodic sentence arises from the prominence which it gives to the main statement.

Four centuries ago, Christopher Columbus, one of the Italian mariners whom the decline of their own republics had put at the service of the world and of adventure, seeking for Spain a westward passage to the Indies as a set-off against the achievements of Portuguese discoverers, lighted on America.
With these hopes and in this belief I would urge you, laying aside all hindrance, thrusting away all private aims, to devote yourselves unswervingly and unflinchingly to the vigorous and successful prosecution of this war.

The other prominent position in the sentence is the beginning. Any element in the sentence, other than the subject, becomes emphatic when placed first.

Deceit or treachery he could never forgive.

So vast and rude, fretted by the action of nearly three thousand years, the fragments of this architecture may often seem, at first sight, like works of nature.
--

A subject coming first in its sentence may be emphatic, but hardly by its position alone. In the sentence,

Great kings worshipped at his shrine,

the emphasis upon kings arises largely from its meaning and from the context. To receive special emphasis, the subject of a sentence must take the position of the predicate.

Through the middle of the valley flowed a winding stream.

The principle that the proper place for what is to be made most prominent is the end applies equally to the words of a sentence, to the sentences of a paragraph, and to the paragraphs of a composition.

Chapter IV

A FEW MATTERS OF FORM

Headings. Leave a blank line, or its equivalent in space, after the title or heading of a manuscript. On succeeding pages, if using ruled paper, begin on the first line.

Numerals. Do not spell out dates or other serial numbers. Write them in figures or in Roman notation, as may be appropriate.

August 9, 1918	Chapter XII
Rule 3	352d Infantry

Parentheses. A sentence containing an expression in parenthesis is punctuated, outside of the marks of parenthesis, exactly as if the expression in parenthesis were absent. The expression within is punctuated as if it stood by itself, except that the final stop is omitted unless it is a question mark or an exclamation point.

I went to his house yesterday (my third attempt to see him), but he had left town.
--

He declares (and why should we doubt his good faith?) that he is now certain of success.

(When a wholly detached expression or sentence is parenthesized, the final stop comes before the last mark of parenthesis.)

Quotations. Formal quotations, cited as documentary evidence, are introduced by a colon and enclosed in quotation marks.

The provision of the Constitution is: "No tax or duty shall be laid on articles exported from any state."

Quotations grammatically in apposition or the direct objects of verbs are preceded by a comma and enclosed in quotation marks.

I recall the maxim of La Rochefoucauld, "Gratitude is a lively sense of benefits to come."
--

Aristotle says, "Art is an imitation of nature."

Quotations of an entire line, or more, of verse, are begun on a fresh line and centred, but not enclosed in quotation marks.

Wordsworth's enthusiasm for the Revolution was at first unbounded:

Bliss was it in that dawn to be alive,
But to be young was very heaven!

Quotations introduced by *that* are regarded as in indirect discourse and not enclosed in quotation marks.

Keats declares that beauty is truth, truth beauty.

Proverbial expressions and familiar phrases of literary origin require no quotation marks.

These are the times that try men's souls.

He lives far from the madding crowd.

The same is true of colloquialisms and slang.

References. In scholarly work requiring exact references, abbreviate titles that occur frequently, giving the full forms in an alphabetical list at the end. As a general practice, give the references in parenthesis or in footnotes, not in the body of the sentence. Omit the words *act*, *scene*, *line*, *book*, *volume*, *page*, except when referring by only one of them. Punctuate as indicated below.

In the second scene of the third act	In III.ii (still better, simply insert III.ii in parenthesis at the proper place in the sentence)
After the killing of Polonius, Hamlet is placed under guard (IV. ii. 14).	
2 Samuel i:17-27	Othello II.iii. 264-267, III.iii. 155-161

Titles. For the titles of literary works, scholarly usage prefers italics with capitalized initials. The usage of editors and publishers varies, some using italics with capitalized initials, others using Roman with capitalized initials and with or without quotation marks. Use italics (indicated in manuscript by underscoring), except in writing for a periodical that follows a different practice. Omit initial *A* or *The* from titles when you place the possessive before them.

The Iliad; the Odyssey; As You Like It; To a Skylark; The Newcomes; A Tale of Two Cities; Dickens's Tale of Two Cities.

Chapter V

WORDS AND EXPRESSIONS COMMONLY MISUSED

(Many of the words and expressions here listed are not so much bad English as bad style, the commonplaces of careless writing. As illustrated under Feature, the proper correction is likely to be not the replacement of one word or set of words by another, but the replacement of vague generality by definite statement.)

All right. Idiomatic in familiar speech as a detached phrase in the sense, "Agreed," or "Go ahead." In other uses better avoided. Always written as two words.

As good or better than. Expressions of this type should be corrected by rearranging the sentence.

My opinion is as good or better than his.	My opinion is as good as his, or better (if not better).
---	--

As to whether. *Whether* is sufficient; see under Rule 13.

Bid. Takes the infinitive without *to*. The past tense is bade.

Case. The *Concise Oxford Dictionary* begins its definition of this word: "instance of a thing's occurring; usual state of affairs." In these two senses, the word is usually unnecessary.

In many cases, the rooms were poorly ventilated.	Many of the rooms were poorly ventilated.
It has rarely been the case that any mistake has been made.	Few mistakes have been made.

See Wood, *Suggestions to Authors*, pp. 68-71, and Quiller-Couch, *The Art of Writing*, pp. 103-106.

Certainly. Used indiscriminately by some speakers, much as others use very, to inten-

34 CHAPTER V. WORDS AND EXPRESSIONS COMMONLY MISUSED

sify any and every statement. A mannerism of this kind, bad in speech, is even worse in writing.

Character. Often simply redundant, used from a mere habit of wordiness.

Acts of a hostile character	Hostile acts
-----------------------------	--------------

Claim, vb. With object-noun, means *lay claim to*. May be used with a dependent clause if this sense is clearly involved: "He claimed that he was the sole surviving heir." (But even here, "claimed to be" would be better.) Not to be used as a substitute for *declare, maintain, or charge*.

Compare. To *compare* is to point out or imply resemblances, between objects regarded as essentially of different order; to *compare with* is mainly to point out differences, between objects regarded as essentially of the same order. Thus life has been compared to a pilgrimage, to a drama, to a battle; Congress may be compared with the British Parliament. Paris has been compared to ancient Athens; it may be compared with modern London.

Clever. This word has been greatly overused; it is best restricted to ingenuity displayed in small matters.

Consider. Not followed by *as* when it means, "believe to be." "I consider him thoroughly competent." Compare, "The lecturer considered Cromwell first as soldier and second as administrator," where "considered" means "examined" or "discussed."

Dependable. A needless substitute for *reliable, trustworthy*.

Due to. Incorrectly used for *through, because of, or owing to*, in adverbial phrases: "He lost the first game, due to carelessness." In correct use related as predicate or as modifier to a particular noun: "This invention is due to Edison;" "losses due to preventable fires."

Effect. As noun, means *result*; as verb, means *to bring about, accomplish* (not to be confused with *affect*, which means "to influence").

As noun, often loosely used in perfunctory writing about fashions, music, painting, and other arts: "an Oriental effect;" "effects in pale green;" "very delicate effects;" "broad effects;" "subtle effects;" "a charming effect was produced by." The writer who has a definite meaning to express will not take refuge in such vagueness.

Etc. Not to be used of persons. Equivalent to *and the rest, and so forth*, and hence not to be used if one of these would be insufficient, that is, if the reader would be left in doubt as to any important particulars. Least open to objection when it represents the last terms of a list already given in full, or immaterial words at the end of a quotation. At the end of a list introduced by *such as, for example*, or any similar expression, *etc.* is incorrect.

Fact. Use this word only of matters of a kind capable of direct verification, not of

matters of judgment. That a particular event happened on a given date, that lead melts at a certain temperature, are facts. But such conclusions as that Napoleon was the greatest of modern generals, or that the climate of California is delightful, however incontestable they may be, are not properly facts.

On the formula *the fact that*, see under Rule 13.

Factor. A hackneyed word; the expressions of which it forms part can usually be replaced by something more direct and idiomatic.

His superior training was the great factor in his winning the match.	He won the match by being better trained.
Heavy artillery is becoming an increasingly important factor in deciding battles.	Heavy artillery is playing a larger and larger part in deciding battles.

Feature. Another hackneyed word; like *factor* it usually adds nothing to the sentence in which it occurs.

A feature of the entertainment especially worthy of mention was the singing of Miss A.	(Better use the same number of words to tell what Miss A. sang, or if the programme has already been given, to tell something of how she sang.)
--	---

As a verb, in the advertising sense of ~~offer as a special attraction~~, to be avoided.

Fix. Colloquial in America for *arrange*, *prepare*, *mend*. In writing restrict it to its literary senses, *fasten*, *make firm* or *immoveable*, etc.

He is a man who. A common type of redundant expression; see Rule 13.

He is a man who is very ambitious.	He is very ambitious.
Spain is a country which I have always wanted to visit.	I have always wanted to visit Spain.

However. ~~In the meaning nevertheless, not to come first in its sentence or clause.~~

The roads were almost impassable. However, we at last succeeded in reaching camp.	The roads were almost impassable. At last, however, we succeeded in reaching camp.
---	--

When however comes first, it means in whatever way or to whatever extent.

However you advise him, he will probably do as he thinks best.	However discouraging the prospect, he never lost heart.
--	---

Kind of. ~~Not to be used as a substitute for *rather* (before adjectives and verbs), or except in familiar style, for *something like* (before nouns).~~ Restrict it to its literal sense: "Amber is a kind of fossil resin;" "I dislike that kind of notoriety." The same holds true of *sort of*.

Less. Should not be misused for *fewer*.

He had less men than in the previous campaign.	He had fewer men than in the previous campaign.
--	---

Less refers to quantity, *fewer* to number. "His troubles are less than mine" means "His troubles are not so great as mine." "His troubles are fewer than mine" means "His troubles are not so numerous as mine." It is, however, correct to say, "The signers of the petition were less than a hundred," where the round number, a hundred, is something like a collective noun, and *less* is thought of as meaning a less quantity or amount.

Line, along these lines. *Line* in the sense of *course of procedure, conduct, thought*, is allowable, but has been so much overworked, particularly in the phrase *along these lines*, that a writer who aims at freshness or originality had better discard it entirely.

Mr. B. also spoke along the same lines.	Mr. B. also spoke, to the same effect.
He is studying along the line of French literature.	He is studying French literature.

Literal, literally. ~~Often incorrectly used in support of exaggeration or violent metaphor.~~

A literal flood of abuse	A flood of abuse
Literally dead with fatigue	Almost dead with fatigue (dead tired)

Lose out. ~~Meant to be more emphatic than lose, but actually less so, because of its commonness.~~ The same holds true of *try out*, *win out*, *sign up*, *register up*. With a

number of verbs, *out* and *up* form idiomatic combinations: *find out*, *run out*, *turn out*, *cheer up*, *dry up*, *make up*, and others, each distinguishable in meaning from the simple verb. *Lose out* is not.

Most. Not to be used for almost.

Most everybody	Almost everybody
Most all the time	Almost all the time

Nature. Often simply redundant, used like *character*.

Acts of a hostile nature	Hostile acts
--------------------------	--------------

Often vaguely used in such expressions as "a lover of nature;" "poems about nature." Unless more specific statements follow, the reader cannot tell whether the poems have to do with natural scenery, rural life, the sunset, the untracked wilderness, or the habits of squirrels.

Near by. Adverbial phrase, not yet fully accepted as good English, though the analogy of *close by* and *hard by* seems to justify it. *Near*, or *near at hand*, is as good, if not better.

Not to be used as an adjective; use *neighboring*.

Oftentimes, oftentimes. Archaic forms, no longer in good use. The modern word is *often*.

One hundred and one. Retain the *and* in this and similar expressions, in accordance with the unvarying usage of English prose from Old English times.

One of the most. Avoid beginning essays or paragraphs with this formula, as, "One of the most interesting developments of modern science is, etc.;" "Switzerland is one of the most interesting countries of Europe." There is nothing wrong in this; it is simply threadbare and forcible-feeble.

People. *The people* is a political term, not to be confused with *the public*. From the people comes political support or opposition; from the public comes artistic appreciation or commercial patronage.

The word *people* is not to be used with words of number, in place of *persons*. If of "six people" five went away, how many "people" would be left?

Phase. Means a stage of transition or development: "the phases of the moon;" "the last phase." Not to be used for *aspect* or *topic*.

Another phase of the subject	Another point (another question)
------------------------------	----------------------------------

Possess. Not to be used as a mere substitute for *have* or *own*.

He possessed great courage.	He had great courage (was very brave).
-----------------------------	--

38 CHAPTER V. WORDS AND EXPRESSIONS COMMONLY MISUSED

He was the fortunate possessor of	He owned
-----------------------------------	----------

Respective, respectively. These words may usually be omitted with advantage.

Works of fiction are listed under the names of their respective authors.	Works of fiction are listed under the names of their authors.
The one mile and two mile runs were won by Jones and Cummings respectively.	The one mile and two mile runs were won by Jones and by Cummings.

In some kinds of formal writing, as in geometrical proofs, it may be necessary to use *respectively*, but it should not appear in writing on ordinary subjects.

So. Avoid, in writing, the use of *so* as an intensifier: "so good;" "so warm;" "so delightful."

On the use of *so* to introduce clauses, see Rule 4.

Sort of. See under *Kind of*.

State. Not to be used as a mere substitute for *say, remark*. Restrict it to the sense of *express fully or clearly*, as, "He refused to state his objections."

Student body. A needless and awkward expression, meaning no more than the simple word *students*.

A member of the student body	A student
Popular with the student body	Liked by the students
The student body passed resolutions.	The students passed resolutions.

System. Frequently used without need.

Dayton has adopted the commission system of government.	Dayton has adopted government by commission.
The dormitory system	Dormitories

Thanking you in advance. This sounds as if the writer meant, "It will not be worth my while to write to you again." Simply write, "Thanking you," and if the favor which you have requested is granted, write a letter of acknowledgment.

They. A common inaccuracy is the use of the plural pronoun when the antecedent is a distributive expression such as *each, each one, everybody, every one, many a man*, which, though implying more than one person, requires the pronoun to be in the singular. Similar to this, but with even less justification, is the use of the plural pronoun with the antecedent *anybody, any one, somebody, some one*, the intention being either to avoid the awkward "he or she," or to avoid committing oneself to either. Some bashful speakers even say, "A friend of mine told me that they, etc."

Use *he* with all the above words, unless the antecedent is or must be feminine.

Very. Use this word sparingly. Where emphasis is necessary, use words strong in themselves.

Viewpoint. Write *point of view*, but do not misuse this, as many do, for *view* or *opinion*.

While. Avoid the indiscriminate use of this word for *and, but, and although*. Many writers use it frequently as a substitute for *and* or *but*, either from a mere desire to vary the connective, or from uncertainty which of the two connectives is the more appropriate. In this use it is best replaced by a semicolon.

40 CHAPTER V. WORDS AND EXPRESSIONS COMMONLY MISUSED

This is entirely correct, as shown by the paraphrase,

The office and salesrooms are on the ground floor, while the rest of the building is devoted to manufacturing.	The office and salesrooms are on the ground floor; the rest of the building is devoted to manufacturing.
--	--

Its use as a virtual equivalent of *although* is allowable in sentences where this leads to no ambiguity or absurdity.

While I admire his energy, I wish it were employed in a better cause.

I admire his energy; at the same time I wish it were employed in a better cause.
--

Compare:

While the temperature reaches 90 or 95 degrees in the daytime, the nights are often chilly.	Although the temperature reaches 90 or 95 degrees in the daytime, the nights are often chilly.
---	--

The paraphrase,

The temperature reaches 90 or 95 degrees in the daytime; at the same time the nights are often chilly,
--

shows why the use of *while* is incorrect.

In general, the writer will do well to use *while* only with strict literalness, in the sense of *during the time that*.

Whom. Often incorrectly used for *who* before *he said* or similar expressions, when it is really the subject of a following verb.

His brother, whom he said would send him the money	His brother, who he said would send him the money
The man whom he thought was his friend	The man who (that) he thought was his friend (whom he thought his friend)

Worth while. Overworked as a term of vague approval and (with *not*) of disapproval. Strictly applicable only to actions: "Is it worth while to telegraph?"

His books are not worth while.	His books are not worth reading (not worth one's while to read; do not repay reading).
The use of <i>worth while</i> before a noun ("a worth while story") is indefensible.	

Would. A conditional statement in the first person requires *should*, not *would*.

I should not have succeeded without his help.
The equivalent of <i>shall</i> in indirect quotation after a verb in the past tense is <i>should</i> , not <i>would</i> .

He predicted that before long we should have a great surprise.
To express habitual or repeated action, the past tense, without <i>would</i> , is usually sufficient, and from its brevity, more emphatic.

Once a year he would visit the old mansion.	Once a year he visited the old mansion.
---	---

Chapter VI

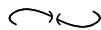
WORDS OFTEN MISSPELLED

accidentally	formerly	privilege
advice	humorous	pursue
affect	hypocrisy	repetition
beginning	immediately	rhyme
believe	incidentally	rhythm
benefit	latter	ridiculous
challenge	led	sacrilegious
criticize	lose	seize
deceive	marriage	separate
definite	mischief	shepherd
describe	murmur	siege
despise	necessary	similar
develop	occurred	simile
disappoint	parallel	too
duel	Philip	tragedy
ecstasy	playwright	tries
effect	preceding	undoubtedly
existence	prejudice	until
fiery	principal	

Write *to-day*, *to-night*, *to-morrow* (but not *together*) with hyphen.

Write *any one*, *every one*, *some one*, *some time* (except the sense of *formerly*) as two words.

THE END



nils m holm
zen style
programming

preface

A program is a description of an abstract, general solution to a specific problem. It is typically written in a formal language called a programming language. The primary purpose of a program is to be understood by fellow human beings, thereby spreading knowledge. In order to achieve maximal readability, a programming language should have certain properties:

1. It should be small and uniform;
2. It should be free from ambiguity;
3. It should provide a high degree of abstraction;
4. It should be independent from concrete computer architectures.

The first points are no-brainers. If a language is too complex or has no uniform syntax and semantics, programmers will have to look up things in the manual perpetually instead of concentrating on the actual problem. If the language introduces ambiguity, people will eventually choose one possible outcome internally and start writing programs that depend on their imagination instead of facts.

A high degree of abstraction means that the language should provide means of dealing with recurring tasks gracefully and without having to reinvent the wheel over and over again. This may seem like a contradiction to 1., but this text will show that this does not have to be the case.

A programming language that is used to describe algorithms in a readable way must be fully architecture-neutral. As soon as the language depends on features of a particular machine, the first principle is violated, because the knowledge that is necessary to understand a program then includes the knowledge of the underlying architecture.

The first part of this book introduces the concept of functional programming, describes a language that fulfills all of the above requirements, and shows how to solve simple problems in it.

Once a language has been chosen, it can be used to formulate solutions to all kinds of logic problems. Programming is limited to solutions of *logic* problems, because it cannot answer questions like “how can we learn to live in peace?” and “why is life worth living?”. These are the limits of science as we know it. The class of problems that *can* be solved by programs is much smaller. It typically involves very clearly defined tasks like

- find permutations of a set;
- find factors of an integer;
- represent an infinite sequence;
- translate formal language A to language B;
- find a pattern in a sequence of characters;
- solve a system of assertions.

Of course these tasks have to be defined in much greater detail in order to be able to solve them programmatically. This is what the second part of the book is about: creating general solutions to logic problems.

The language that will be used in this book is a minimalistic variant of Scheme called `zenlisp`. Its only data types are symbols and ordered pairs. Nothing else is required to describe algorithms for

solving all of the logic problems listed above. The language may appear useless to you, because it does not provide any access to “real-world” application program interfaces and it cannot be used to write interactive programs, but I have chosen this language on purpose: it demonstrates that programming does not depend on these features.

The second part of the book shows how to apply the techniques of the first part to some problems of varying complexity. The topics discussed in this part range from simple functions for sorting or permuting lists to regular expression matching, formal language translation, and declarative programming. It contains the full source code to a source-to-source compiler, a meta-circular interpreter and a logic programming system.

The third part, finally, shows how to implement the abstraction layer that is necessary for solving problems in an abstract way on a concrete computer. It reproduces the complete and heavily annotated source code for an interpreter of `zenlisp`.

The first chapter of this part strives to deliver an example of readable code in a language that is not suitable for abstraction. It attempts to develop a programming style that does not depend on annotations to make the intention of the programmer clear. Comments are interspersed between functions, though, because prose is still easier to read than imperative code.

At this point the tour ends. It starts with an abstract and purely symbolic view on programming, advances to the application of symbolic programming to more complex problems, and concludes with the implementation of a symbolic programming system on actual computer systems.

I hope that you will enjoy the tour!

Nils M Holm, September 2008

contents

part one: symbolic programming	9
1. basic aspects	9
1.1 symbols and variables	9
1.2 functions	10
1.2.1 calling functions	11
1.2.2 function composition	12
1.3 conditions	13
1.4 recursion	15
1.5 forms and expressions	16
1.5.1 lists	17
1.5.2 forms	19
1.5.3 expressions	20
1.6 recursion over lists	20
2. more interesting aspects	24
2.1 variadic functions	24
2.2 equality and identity	26
2.2.1 comparing more complex structures	27
2.3 more control	29
2.4 structural recursion	30
2.5 functions revisited	33
2.5.1 bound and free variables	34
2.6 local contexts	35
2.6.1 closures	36
2.6.2 recursive functions	38
2.6.3 recursive closures	39
2.6.4 recursive bindings	40
2.7 higher-order functions	42
2.7.1 mapping	43
2.7.2 folding	45
3. rather esoteric aspects	47
3.1 numeric functions	47
3.1.1 numeric predicates	48
3.1.2 integer functions	50
3.1.3 rational functions	51
3.1.4 type checking and conversion	52
3.2 side effects	54
3.2.1 subtle side effects	55
3.2.2 evaluation	56
3.3 metaprogramming	56
3.3.1 programs hacking programs	57

3.3.2	beta reduction by substitution	59
3.4	packages	62

part two: algorithms 63

4.	list functions	63
4.1	heads and tails	63
4.2	find the n'th tail of a list	64
4.3	count the atoms of a form	64
4.4	flatten a tree	65
4.5	partition a list	66
4.6	folding over multiple lists	67
4.7	substitute variables	68
5.	sorting	69
5.1	insertion sort	69
5.2	quicksort	70
5.3	mergesort	72
5.4	unsorting lists	73
6.	logic and combinatoric functions	78
6.1	turning lists into sets	78
6.2	union of sets	78
6.3	find members with a given property	79
6.4	verify properties	80
6.5	combinations of sets	80
6.6	permutations of sets	83
7.	math functions	86
7.1	sequences of numbers	86
7.2	fast factorial function	86
7.3	integer factorization	88
7.4	partitioning integers	89
7.5	exploring the limits of computability	91
7.6	transposing matrixes	93
8.	data structures	94
8.1	generators	94
8.2	streams	95
8.3	ml-style records	99
9.	compilers	106
9.1	translating infix to prefix	106
9.1.1	formal grammars	106
9.1.2	left versus right recursion	111
9.1.3	implementation	113

9.2	translating prefix to infix	117
9.3	regular expressions	121
9.3.1	regular expression compilation	123
9.3.2	regular expression matching	126
9.4	meta-circular interpretation	129
10.	mexprc – an m-expression compiler	139
10.1	specification	139
10.1.1	annotated grammar	141
10.2	implementation	143
10.2.1	lexical analysis	144
10.2.2	syntax analysis and code synthesis	147
10.3	example programs	160
10.3.1	append lists	161
10.3.2	the towers of hanoi	162
10.3.4	n queens	163
11.	another micro kanren	165
11.1	introduction	165
11.1.1	functions versus goals	165
11.1.2	unification	166
11.1.3	logic operators	168
11.1.4	parameterized goals	170
11.1.5	reification	171
11.1.6	recursion	171
11.1.7	converting predicates to goals	173
11.1.8	converting functions to goals	174
11.1.9	cond versus any	175
11.1.10	first class variables	176
11.1.11	first class goals	178
11.1.12	negation	179
11.1.13	cutting	180
11.2	a logic puzzle	182
11.3	implementation	186
11.2.1	basics	186
11.2.2	goals	188
11.2.3	interface	190
part three: zenlisp implementation		195
12.	c part	195
12.1	prelude and data declarations	195
12.2	miscellaneous functions	204
12.3	error reporting	205
12.4	counting functions	207
12.5	memory management	208

12.6	symbol tables	213
12.7	reader	216
12.8	primitive operation handlers	222
12.9	special form handlers	229
12.10	evaluator	247
12.11	printer	255
12.12	initialization	258
12.13	interpreter interface	260
12.14	interpreter shell	264
13.	lisp part	270
13.1	base library	270
13.2	iterator package	274
13.3	natural math functions	274
13.4	integer math functions	286
13.5	rational math functions	293

appendix 303

A.1	tail call rules	303
A.2	zenlisp functions	304
A.2.1	definitions	304
A.2.2	control	305
A.2.3	lists	305
A.2.4	miscellanea	306
A.2.5	packages	307
A.2.6	meta functions	307
A.3	math functions	308
A.4	working with zenlisp	309
A.4.1	the development cycle	311
A.5	zenlisp for the experienced schemer	314
A.6	answers to some questions	314
A.7	list of figures	324
A.8	list of example programs	324
A.9	code license	326

index 327

part one symbolic programming

This part discusses the foundations of symbolic programming by means of a purely symbolic, lexically scoped, functional variant of LISP called `zenlisp`.

`Zenlisp` is similar to Scheme, but simpler.

Being purely symbolic, the language has only two fundamental data types: the symbol and the ordered pair.

`Zenlisp` implements the paradigm of *functional programming*. Functional programming focuses on the evaluation of expressions. Programs are sets of functions that map values to values. Tail-recursive expressions evaluate in constant space, making iteration a special case of recursion.

`Zenlisp` programs are typically free of side effects.

Although the techniques described in this text are presented in a LISPy language, the purely symbolic approach makes it easy to adapt those techniques to other functional languages and maybe even to languages of other paradigms.

1. basic aspects

1.1 symbols and variables

This is a *quoted symbol*:

```
'marmelade
```

It is called a *quoted symbol* because of the quote character in front of it.

Anything that is quoted *reduces to* itself:

```
'marmelade => 'marmelade
```

The `=>` operator reads “*reduces to*” or “*evaluates to*”.

The lefthand side of `=>` is a program and its righthand is called the *normal form* (or the “result”) of that program.

A symbol that is *not* quoted is called a *variable*.

Here is a variable:

```
food
```

The normal form of a variable is the value associated with that variable:

zen style programming

```
(define food 'marmelade)
food => 'marmelade
```

The above **define** binds the value `'marmelade` to the variable *food*.

After binding a variable to a value, each reference to the variable results in the value bound to that variable; the variable reduces to its value:

```
food => 'marmelade
```

Symbols and variables are independent from each other:

```
(define marmelade 'fine-cut-orange)
marmelade => 'fine-cut-orange
```

While *marmelade* now refers to `'fine-cut-orange`, the quoted symbol `'marmelade` still reduces to itself and *food* still reduces to `'marmelade`:

```
'marmelade => 'marmelade
food => 'marmelade
```

Once defined, the value of a variable normally does not change.
--

A symbol that has no value associated with it is not a valid variable:

```
undefined-symbol => bottom
```

Bottom denotes an *undefined value*.

Anything that reduces to bottom should be considered an error.

A quoted symbol is its own value, so a quoted symbol without an association is fine:

```
'undefined-symbol => 'undefined-symbol
```

1.2 functions

The following *expression* applies the **append** function to two arguments:

```
(append '(pizza with) '(extra cheese))
=> '(pizza with extra cheese)
```

The data `'(pizza with)` and `'(extra cheese)` are *lists*. **Append** appends them.

Lists are in their normal forms because they are quoted:

```
'(pizza with cheese) => '(pizza with cheese)
```

The normal form of a function application is the value returned by the applied function:

```
(reverse '(pizza with pepperonies))
=> '(pepperonies with pizza)
```

Lists and function applications share the same notation; they differ only by the quote character that is attached to lists:

```
(reverse '(ice water)) => '(water ice)
'(reverse '(ice water)) => '(reverse '(ice water))
```

Attaching a quote character to a function application turns it into a list, but...

Removing a quote from a list does not necessarily turn it into a function application:

```
(pizza with pepperonies) => bottom
```

This does not work, because no function with the name “pizza” has been defined before.

You can define one, though:

```
(define (pizza topping)
  (list 'pizza 'with topping))
```

defines a function named *pizza* that takes one argument named *topping*.

The *body* of the function is an application of **list** to three arguments: the symbols `'pizza` and `'with` and the variable *topping*. The body of a function is sometimes also called the *term* of that function.

When *pizza* is applied, **list** forms a new list containing the given arguments:

```
(pizza 'anchovies) => '(pizza with anchovies)
```

BTW, the quotes of the symbols `'pizza` and `'with` have not vanished.

A quote character in front of a list quotes everything contained in that list:

```
'(gimme a (pizza 'hot-chili))
=> '(gimme a (pizza 'hot-chili))
```

If you want to reduce the *members* of a list, you have to use the **list** function:

```
(list 'gimme 'a (pizza 'hot-chili))
=> '(gimme a (pizza with hot-chili))
```

1.2.1 calling functions

Pizza-2 is like *pizza*, but accepts two arguments:

```
(define (pizza-2 top1 top2)
  (list 'pizza 'with top1 'and top2))
```

The variables *top1* and *top2* are the *variables* of the function. They are sometimes also called its *formal arguments*.

zen style programming

The values the function is applied to are called *actual arguments* or just *arguments*. A list of variables of a function is called its *argument list*.

In the following example, `'(extra cheese)` and `'(hot chili)` are (actual) arguments:

```
(pizza-2 '(extra cheese) '(hot chili))  
=> '(pizza with (extra cheese) and (hot chili))
```

Here is what happens during the above function application:

1. the values of *top1* and *top2* are saved;
2. `'(extra cheese)` is bound to *top1*;
3. `'(hot chili)` is bound to *top2*;
4. `(list 'pizza 'with top1 'and top2)` is reduced, giving a result *R*;
5. *top1* and *top2* are bound to the values saved in 1.;
6. *R* is returned.

Because of 1. and 5., the variables of a function are *local* to that function.

Therefore multiple functions may share the same variable names. Each of these function has its own local *x*:

```
(define (f x) (append x x))  
(define (g x) (reverse x))
```

Arguments of functions are matched by position:

```
(pizza-2 'olives 'pepper) => '(pizza with olives and pepper)  
(pizza-2 'pepper 'olives) => '(pizza with pepper and olives)
```

1.2.2 function composition

This is a composition of the functions **reverse** and **append**:

```
(reverse (append '(ice with) '(juice orange)))  
=> '(orange juice with ice)
```

Function composition is used to form new functions from already existing ones:

```
(define (palindrome x)  
  (append x (reverse x)))
```

In *palindrome*, the second argument of **append** is the value returned by **reverse**.

Zenlisp uses *applicative order evaluation*.

Therefore `(reverse x)` is first reduced to its normal form and that normal form is passed as an argument to **append**.

Multiple arguments to the same function are not reduced in any specific order, but in the examples

left-to-right evaluation is assumed.

Here is a sample application of *palindrome*:

```
(palindrome '#12345)
```

The datum `'#12345` is just a `zenlisp` short cut for `'(1 2 3 4 5)` — every list of single-character symbols can be abbreviated this way.

The application reduces as follows (the `->` operator denotes a *partial reduction*):

```
(palindrome '#12345)
-> (append x (reverse x))           ; reduced palindrome
-> (append '#12345 (reverse x))     ; reduced first x
-> (append '#12345 (reverse '#12345)) ; reduced second x
-> (append '#12345 '#54321)         ; reduced application of reverse
=> '#1234554321                    ; reduced application of append
```

In a partial reduction, one or multiple sub-expressions of a function application (like variables or embedded applications) are reduced to their normal forms.

1.3 conditions

The symbol `:f` denotes logical falsity:

```
:f => :f
```

`:F` reduces to itself, so it does not have to be quoted.

The symbols `:t` and `t` denote logical truth:

```
:t => :t
t => :t
```

`:T` and `t` both reduce to `:t`, so they do not have to be quoted either.

In case you wonder why there are *two* values representing truth, `t` simply looks better than `:t` in some contexts.

The **cond** *pseudo function* implements *conditional reduction*:

```
(cond (:f 'bread)
      (:t 'butter))
=> 'butter
```

Each argument of **cond** is called a *clause*.

Each clause consists of a *predicate* and a *body*:

```
(predicate body)
```

Cond works as follows:

zen style programming

1. it reduces the predicate of its first clause to its normal form;
2. if the predicate reduces to truth, the value of **cond** is the normal form of the body associated with that predicate;
3. if the predicate reduces to falsity, **cond** proceeds with the next clause.

Cond returns as soon as a true predicate is found:

```
(cond (:f 'bread)
      (:t 'butter)
      (:t 'marmelade))
=> 'butter
```

Therefore, the above **cond** will *never* return 'marmelade.

It is an error for **cond** to run out of clauses:

```
(cond (:f 'false)
      (:f 'also-false))
=> bottom
```

Therefore, the last clause of **cond** should always have constant truth as its predicate, so it can catch all remaining cases.

Here is a function that uses **cond**:

```
(define (true value)
  (cond (value (list value 'is 'true))
        (t (list value 'is 'false))))
```

True finds out whether a value is “true” or not.

Let us try some values:

```
(true :f) => '(:f is false)
```

As expected.

```
(true :t) => '(:t is true)
```

Also fine. Here are the truth values of some other expressions:¹

```
(true 'orange-fine-cut) => '(orange-fine-cut is true)
  (true '0) => '(0 is true)
  (true true) => '({closure (value)} is true)
  (true ()) => '(() is true)
```

It seems that most values thrown at *true* turn out to be true.

Indeed:

Cond interprets all values except for :f as truth. Only :f is false.

¹ We will investigate at a later time why *true* reduces to {closure (value)}.

1.4 recursion

Here are some interesting functions:

Car extracts the first member of a list:

```
(car '(first second third)) => 'first
(car '#abcdef) => 'a
```

The **cdr** function extracts the tail of a list:

```
(cdr '(first second third)) => '(second third)
(cdr '#abcdef) => '#bcdef
```

The tail of a list of one member is **()**:

```
(cdr '(first)) => ()
```

() is pronounced “nil”; it represents the *empty list*.

The **null** predicate tests whether its argument is **()**:

```
(null '#abcde) => :f
(null ()) => :t
```

A function is called a predicate when it *always* returns either **:t** or **:f**.

The **eq** predicate tests whether two symbols are identical:

```
(eq 'orange 'orange) => :t
(eq 'orange 'apple) => :f
```

Memq makes use of all of the above functions:

```
(define (memq x a)
  (cond ((null a) :f)
        ((eq x (car a)) a)
        (t (memq x (cdr a)))))
```

It locates the first occurrence of *x* in the list of symbols *a*:

```
(memq 'c '#abcde) => '#cde
```

When *a* does not contain *x*, **memq** returns **:f**:

```
(memq 'x '#abcde) => :f
```

When the list passed to **memq** is empty, the first clause of the **cond** of **memq** applies and **memq** returns **:f**:

```
(memq 'x ()) => :f
```

The second clause uses **eq** to find out whether *x* and **(car a)** denote the same symbol. If so,

zen style programming

memq returns *a*:

```
(memq 'x '#x) => '#x
```

The last clause applies **memq** to the *tail* of the list *a*:

```
(memq x (cdr a))
```

Because **memq** applies **memq** in order to compute its own result, it is said to *recurse*; **memq** is called a *recursive function*.

An application of **memq** reduces as follows:

```
(memq 'c '#abcde)
-> (cond ((null '#abcde) :f)
        ((eq 'c (car '#abcde)) '#abcde)
        (t (memq 'c (cdr '#abcde)))))
-> (cond ((eq 'c (car '#abcde)) '#abcde)
        (t (memq 'c (cdr '#abcde)))))
-> (cond (t (memq 'c (cdr '#abcde)))))
-> (memq 'c (cdr '#abcde))
-> (memq 'c '#bcde)
-> (memq 'c '#cde)
=> '#cde
```

Each clause of **cond** covers one *case*.

The non-recursive cases of a recursive function are called its *trivial cases*, the recursive cases are called its *general cases*.

Recursion is used to express iteration.
--

1.5 forms and expressions

Car and **cdr** extract the *head* (the first member) and the *tail* (all members but the first) of a list:

```
(car '(large banana split)) => 'large
(cdr '(large banana split)) => '(banana split)
```

The **cons** function creates a fresh list by attaching a new head to an existing list:

```
(cons 'banana ()) => '(banana)
(cons 'banana '(split)) => '(banana split)
```

However, the second argument of **cons** does not have to be a list:

```
(cons 'heads 'tails) => '(heads . tails)
```

A structure of the form

```
(car-part . cdr-part)
```

is called a *dotted pair*.

The functions **cons**, **car**, and **cdr** are correlated in such a way that

```
(cons (car x) (cdr x)) = x
```

holds for any pair *x*.

The car part and cdr part of each pair may be another pair:

```
((caar . cdar) . (cdar . cddr))
```

The name “caar” denotes the “car part of a car part”, “cdar” denotes the “cdr part of a car part”, etc.

There is a set of equally named functions that extract these parts from nested pairs:

```
(caar '((caar . cdar) . (cadr . cddr))) => 'caar  
(cdar '((caar . cdar) . (cadr . cddr))) => 'cdar  
(cadr '((caar . cdar) . (cadr . cddr))) => 'cadr  
(cddr '((caar . cdar) . (cadr . cddr))) => 'cddr
```

Zenlisp provides functions to extract data from up to four levels of nested pairs.

For instance **cddddr** extracts the **cdr**⁴ part (the tail starting at the fourth member of a list) and **caddr** returns the “car of the cdr of the cdr” of a datum (which happens to be the second member of a list):

```
(cddddr '#abcdef) => '#ef  
(caddr '#abcdef) => 'c
```

Hint: To decode functions for accessing nested pairs, read the a’s and d’s in their names backward:

```
(cadadr '(a (b c) d)) ; remove last 'd', do cdr  
-> (cadar '((b c) d)) ; remove last 'a', do car  
-> (cadr '(b c))      ; remove last 'd', do cdr  
-> (car '(c))         ; do final car  
=> 'c
```

1.5.1 lists

A *list* is

1. either the empty list **()**;
2. or a pair whose cdr part is a *list*.

The **listp** predicate tests whether a datum is a list:²

² Appending a “p” to the name of a predicate is ancient LISP tradition, but zenlisp follows this tradition in a rather liberal way.

zen style programming

```
(define (listp x)
  (cond ((null x) :t)
        ((atom x) :f)
        (t (listp (cdr x)))))
```

Listp uses the **atom** predicate, which tests whether its argument is *atomic*. Anything that cannot be split (by **car** or **cdr**) is atomic:

```
(car 'symbol) => bottom
(car ()) => bottom
(cdr 'symbol) => bottom
(cdr ()) => bottom
```

Let us apply **listp** to some data:

```
(listp ()) => :t
(listp '(banana split)) => :t
(listp '#abcde) => :t
```

Fine, and now some negatives:

```
(listp 'banana) => :f
(listp '(heads . tails)) => :f
```

'Banana is obviously not a list, nor is the tail of '(heads . tails).

How about these:

```
(listp '(define (f x) x)) => :t
(listp '(x ((y . z)) ())) => :t
```

Yes, lists may contain pairs and lists, and they may be nested to any level.

And this?

```
(listp (append '#abcde 'x)) => :f
```

Whatever appending a symbol to a list gives, it is not a list:

```
(append '#abcde 'x) => '(a b c d e . x)
```

The resulting structure is a hybrid of a dotted pair and a list; it is called a *dotted list*. (Sometimes it is also called an *improper list*, because “proper” lists end with **()**.)

Because lists are pairs, the functions **caar...cdddr** can be used to extract members of lists, too.

For instance:

```
(caar '((first) (second) (third))) => 'first
(cadr '((first) (second) (third))) => '(second)
(cdar '((first) (second) (third))) => '()
(cddr '((first) (second) (third))) => '((third))
```

```
(caddr '((first) (second) (third))) => '(third)
(cdddr '((first) (second) (third))) => ()
(caaddr '((first) (second) (third))) => 'third
```

1.5.2 forms

Each form is either a symbol or a pair.
--

These are forms:

```
marmelade
cdr
:f
(heads . tails)
(banana ; this is a comment
      split)
(define (f x) (f (f x)))
(1 2 3 4 . 5)
#hello-world
```

A *comment* can be placed anywhere inside of a form (but *not* inside of a symbol name!) using a semicolon.

A comment extends up to the end of the current line.

For some forms, there are different notations.

Lists may be expanded to pairs:

```
(large banana split) = (large . (banana . (split . ())))
```

There is no really good reason to do so, but it shows that

Every list is a pair. (But not every pair is a list.)
--

Lists that consist of single-character symbols exclusively may be *condensed*:

```
(h e l l o - w o r l d !) = #hello-world!
```

Condensed forms are useful because they are easier to comprehend, easier to type, and save space.

Did you notice that no form in this subsection was *quoted*?

This is because “form” is an abstract term.

A form is turned into a *datum* by applying the **quote** pseudo function to it:

```
(quote (banana split)) => '(banana split)
```

But you do not have to use **quote** each time you want to create a datum, because

```
'(banana split) => '(banana split)
```

zen style programming

1.5.3 expressions

An expression is a form with a meaning.

In *zenlisp*, there is no difference between *expressions* and *programs*.

Every expression is a program and every program consists of one or multiple expressions.

```
(car '(fruit salad))
```

is a program that extracts the first member of a specific list.

```
(define (palindrome x)
  (append x (reverse x)))
```

is an expression with a *side effect*.

A side effect causes some state to change.

The side effect of **define** is to create a *global definition* by binding a variable to a value. In the above case, the value is a function.

The definition is called “global”, because its binding does not occur inside of a specific function.

Define does have a result, but it is mostly ignored:

```
(define (pizza x) (list 'pizza 'with x)) => 'pizza
```

Because **define** is called only for its side effect, it is called a *pseudo function* or a *keyword*.

Another property of pseudo functions is that they are called *by name*, i.e. their arguments are not reduced before the function is applied.

This is why the clauses of **cond** and the arguments of **quote** and **define** do not have to be quoted:

```
(cond (:f 'pizza) (t 'sushi))
(quote (orange juice))
(define (f x) (f f x))
```

Each expression is either a variable or a (pseudo) function application.
--

1.6 recursion over lists

Here is **reverse**:

```
(define (reverse a)
  (cond ((null a) ())
        (t (append (reverse (cdr a))
                     (list (car a))))))
```

20

Reverse reverses a list.

The trivial case handles the empty list, which does not have to be reversed at all.

The general case reverses the *cdr part* (the rest) of the list and then appends a list containing the *car part* (first member) of the original list to the result.

This is how **reverse** works:

```
(reverse '#abc)
-> (append (reverse '#bc) (list 'a))
-> (append (append (reverse '#c) (list 'b)) (list 'a))
-> (append (append (append (reverse ()) (list 'c)) (list 'b))
      (list 'a))
-> (append (append (append () (list 'c)) (list 'b)) (list 'a))
-> (append (append '#c (list 'b)) (list 'a))
-> (append '#cb (list 'a))
=> '#cba
```

Each member of the argument of **reverse** adds one application of **append**.

This is called *linear recursion*.

In many cases, linear recursion can be avoided by adding an additional argument that carries an intermediate result.

Reverse can be modified in such a way:

```
(define (reverse2 a r)
  (cond ((null a) r)
        (t (reverse2 (cdr a)
                      (cons (car a) r)))))

(define (reverse a) (reverse2 a ()))
```

Reverse is now a “wrapper” around *reverse2*, and *reverse2* does the actual work:

```
(reverse2 '#abc ())
-> (reverse2 '#bc '#a)
-> (reverse2 '#c '#ba)
-> (reverse2 () '#cba)
=> '#cba
```

Because the intermediate result does not grow during the reduction of *reverse2*, the function is said to *reduce in constant space*.

This is achieved by rewriting the recursive application of **reverse** as a *tail call*.

A function *call* is the same as a function application.

A tail call is a function call in a *tail position*.

zen style programming

In the expression

```
(append (reverse (cdr a)) (list (car a)))
```

reverse is *not* in a tail position, because **append** is called when **reverse** returns.

In the expression

```
(reverse2 (cdr a) (cons (car a) r))
```

reverse2 is in a tail position, because *reverse2* is the last function called in the expression.

BTW, **cond** does not count, because

```
(cond (t (reverse2 (cdr a) (cons (car a) r))))
```

can be rewritten as

```
(reverse2 (cdr a) (cons (car a) r))
```

So:

1. the outermost function in a function body is in a tail position;
2. the outermost function in a **cond** body is in a tail position.

A function that uses recursion in tail positions exclusively is called a *tail-recursive* function.

Tail recursion is more efficient than linear recursion.
--

Here is another recursive function, *append2*:

```
(define (append2 a b)
  (cond ((null a) b)
        (t (cons (car a)
                   (append2 (cdr a) b)))))
```

It is called *append2*, because it accepts two arguments. **Append** accepts any number of them:

```
(append '#he '#llo '#- '#wor '#ld) => '#hello-world
```

But this is not the worst thing about *append2*.

Append2 conses (**car a**) to the result of an application of *append2*, so it is not tail-recursive.

Can you write a tail-recursive version of *append2*?

Here it is:

```
(define (r-append2 a b)
  (cond ((null a) b)
        (t (r-append2 (cdr a)
                        (cons (car a) b)))))
```

```
(define (append2 a b)
  (r-append2 (reverse a) b))
```

And this is how it works:

```
(append2 '#abc '#def)
-> (r-append (reverse '#abc) '#def)
-> (r-append '#cba '#def)
-> (r-append '#ba '#cdef)
-> (r-append '#a '#bcdef)
-> (r-append () '#abcdef)
=> '#abcdef
```

Are there any functions that cannot be converted to tail-recursive functions?

Yes, there are. You will see some of them in the following chapter.

2. more interesting aspects

2.1 variadic functions

Here is *intersection*:

```
(define (intersection a b)
  (cond ((null a) ())
        ((memq (car a) b)
         (cons (car a)
                (intersection (cdr a) b)))
        (t (intersection (cdr a) b))))
```

It computes the intersection of two sets of symbols:

```
(intersection '#abcd '#cdef) => '#cd
(intersection '#abcd '#wxyz) => ()
```

If you want to form the intersection of more than two sets, you have to compose applications of *intersection*:

```
(define (intersection3 a b c)
  (intersection a (intersection b c)))
```

To process a variable number of sets, you can pass the sets to a function inside of a list. This is how *intersection-list* works:

```
(define (intersection-list a*)
  (cond ((null a*) a*)
        ((null (cdr a*)) (car a*))
        (t (intersection (car a*)
                           (intersection-list (cdr a*))))))
```

Intersection-list forms the intersection of all sets contained in a list:

```
(intersection-list '()) => ()
(intersection-list '(#abcd)) => '#abcd
(intersection-list '(#abcd #bcde)) => '#bcd
(intersection-list '(#abcd #bcde #cdef)) => '#cd
```

Here is the code of **list**:

```
(define (list . x) x)
```

Yes, that's all. Really.

List is a *variadic function*, a function that accepts a variable number of arguments.

The dot in front of the variable (*x*) of **list** says: “bind a list containing all actual arguments to that variable”:

```
(list 'orange) => '(orange)
(list 'orange 'juice) => '(orange juice)
(list 'orange 'juice 'with 'ice) => '(orange juice with ice)
```

Except for being variadic, **list** is an ordinary function.

Because arguments are reduced to their normal forms *before* they are passed to **list**, lists can include dynamic values:

```
(list (cons 'heads 'tails) (intersection '#abcde '#cdefg))
=> '((heads . tails) #cde)
```

When no arguments are passed to **list**, *x* is bound to a list of no arguments:

```
(list) => ()
```

Here is a version of *intersection-list* that accepts a variable number of sets instead of a list of sets:

```
(define (intersection* . a*)
  (cond ((null a*) a*)
        ((null (cdr a*)) (car a*))
        (t (intersection
              (car a*)
              (apply intersection* (cdr a*))))))
```

There are two differences between *intersection-list* and *intersection**:

1. *intersection** takes a variable number of arguments;
2. *intersection** uses **apply** to recurse.

Apply applies a function to a list of arguments:

```
(apply fun (list arg1 ... argn)) = (fun arg1 ... argn)
```

Here are some examples:

```
(apply cons '(heads tails)) => '(heads . tails)
(apply intersection* '(#abc #bcd)) => '#bc
(apply intersection* (cdr '(#abc #bcd))) => '#bcd
```

Apply can be used to apply a function to a dynamically generated list of arguments.

In *intersection**, it applies the function to the tail of the argument list:

```
(intersection* '#abc '#bcd '#cde)
-> (intersection '#abc (apply intersection*
                             (cdr '(#abc #bcd #cde))))
-> (intersection '#abc (apply intersection* '(#bcd #cde)))
-> (intersection '#abc (intersection '#bcd
                                     (apply intersection* '(#cde))))
-> (intersection '#abc (intersection '#bcd '#cde))
-> (intersection '#abc '#cd)
=> '#c
```


zen style programming

BTW, **apply** can safely be used in tail calls.

Can you write a tail-recursive version of *intersection**?

No solution is provided at this point.

This function creates a non-empty list:

```
(define (non-empty-list first . rest)
  (cons first rest))
```

When applied to some actual arguments, it behaves in the same way as **list**:

```
(non-empty-list 'a 'b 'c) => '#abc
(non-empty-list 'a 'b)   => '#ab
(non-empty-list 'a)      => '#a
```

Applying it to no arguments at all is undefined, though:

```
(non-empty-list) => bottom
```

This is because *non-empty-list* expects *at least one* argument.

There must be one argument for each variable in front of the dot of its dotted argument list:

```
(non-empty-list first . rest)
```

First is bound to the first argument and *rest* is bound to the list of “remaining” argument, if any.

If there is exactly one argument, *rest* is bound to **()**.

There may be any number of arguments in front of the dot:

```
(define (skip-3 dont-care never-mind ignore-me . get-this) get-this)
```

2.2 equality and identity

All symbols are unique.

Therefore all symbol names that are equal refer to the same symbol:

```
marmelade marmelade marmelade marmelade marmelade
```

All these names refer to the same symbol named “marmelade”.

Two instances of the same symbol are called *identical*.

Identity is expressed using the **eq** predicate:

```
(eq 'marmelade 'marmelade) => :t
(eq 'fine-cut 'medium-cut) => :f
```

There is only one *empty list* in `zenlisp`, so all instances of `()` are identical, too:

```
(eq () ()) => :t
```

But **eq** can do more than this:

```
(eq 'symbol '(a . pair)) => :f  
(eq 'symbol '(some list)) => :f
```

When one argument of **eq** is neither a symbol nor `()` and the other one is either a symbol or `()`, **eq** is guaranteed to reduce to falsity.

This may be considered as a means of “built-in type checking”.

So two forms are identical, if...

- they denote the same symbol;
- they are both `()`.

Two forms are *not* identical, if one of them is a pair and the other is not a pair.

When both arguments of **eq** are pairs, the result is *undefined*:

```
(eq '(a . b) '(a . b)) => bottom  
(eq '#abcdef '#abcdef) => bottom
```

Note that “undefined” in this case means that the result is totally unpredictable. The application of **eq** to two equal-looking pairs can yield `:t` at one time and `:f` at another.

Therefore:

Never apply eq to two pairs.

2.2.1 comparing more complex structures

These two lists may or may not be identical:

```
'(bread with butter and marmelade)  
'(bread with butter and marmelade)
```

But a quick glance is enough to find out that both lists contain identical symbols at equal positions, so they could be considered *equal*.

What about these lists:

```
'(bread with (butter and marmelade))  
'((bread with butter) and marmelade)
```

Although the lists contain identical symbols, you would certainly not consider them equal, because they contain different sublists.

zen style programming

Here is a better approach:

Two forms are equal if...

- they are both the same symbol;
- they are both `()`;
- they are both pairs and contain equal car and cdr parts.

The **equal** function tests whether two forms are equal:

```
(define (equal a b)
  (cond ((atom a) (eq a b))
        ((atom b) (eq a b))
        ((equal (car a) (car b))
         (equal (cdr a) (cdr b)))
        (t :f)))
```

Equal returns truth whenever **eq** returns truth:

```
(equal 'fine-cut 'fine-cut) => :t
(equal () ()) => :t
```

In addition, it returns truth when applied to two pairs that *look equal*:

```
(equal '(bread (with) butter)
       '(bread (with) butter)) => :t
```

Because it recurses into the car and cdr parts of its arguments, it detects differences even in nested lists:

```
(equal '(bread (with) butter)
       '(bread (without) butter)) => :f
```

Because **equal** makes sure that **eq** is only applied to arguments that do not cause undefined results, it can be applied safely to any type of datum.

Use eq to express identity and equal to express equality.

Here is **member**:

```
(define (member x a)
  (cond ((null a) :f)
        ((equal x (car a)) a)
        (t (member x (cdr a)))))
```

Member is similar to **memq** [page 15], but uses **equal** instead of **eq**.

Therefore it can find members that **memq** cannot find:

```
(memq '(with) '(bread (with) butter)) => bottom
(member '(with) '(bread (with) butter)) => '((with) butter)
```

2.3 more control

This is how the **or** pseudo function works:

```
(or 'sushi 'pizza 'taco) => 'sushi
(or :f :f 'taco :f) => 'taco
(or :f :f :f) => :f
```

It returns the normal form of the first one of its arguments that does not reduce to falsity, or falsity if all of its arguments reduce to falsity.

Or can be expressed using **cond**:

```
(or a b)      = (cond (a a) (t b))
(or a b c)    = (cond (a a) (b b) (t c))
```

From this equivalence follows that

```
(or a) = (cond (t a)) = a
```

In addition, applying **or** to zero arguments yields the neutral element of the logical “or”:

```
(or) => :f
```

This is how the **and** pseudo function works:

```
(and 'tomato 'lettuce 'bacon) => 'bacon
(and 'tomato :f 'bacon) => :f
```

It returns the first one of its arguments that does not reduce to truth, or the normal form of its last argument if all of them reduce to truth.

And can be expressed using **cond**:

```
(and a b)      = (cond (a b) (t :f))
(and a b c)    = (cond (a (cond (b c)
                                (t :f)))
                        (t :f))
```

In addition:

```
(and a) = a
```

Applying **and** to zero arguments yields the neutral element of the logical “and”:

```
(and) => :t
```

And and **or** implement so-called *short circuit boolean reduction*.

Both of them stop evaluating their arguments as soon as they find a true or false value respectively:

zen style programming

```
(and :f (bottom)) => :f  
(or :t (bottom)) => :t
```

Bottom is a function that evaluates to bottom. Its result is undefined for any arguments passed to it:

```
(bottom) => bottom  
(bottom 'foo) => bottom  
(bottom 'foo (list 'bar)) => bottom
```

Because of a principle known as *bottom preservation*, each function that takes a bottom argument must itself reduce to bottom:

```
(atom (bottom)) => bottom  
(eq 'x (bottom)) => bottom  
(pizza (bottom)) => bottom
```

However, **and** and **or** are *pseudo functions* and their arguments are passed to them in *unreduced* form.

Because

- **and** never reduces any arguments following a “false” one
- and
- **or** never reduces any arguments following a “true” one,

bottom preservation does not apply, and so:

```
(and :f (bottom)) => :f  
(or :t (bottom)) => :t
```

Bottom preservation does not apply to and, or, and cond.

By using **and** and **or**, two clauses of **equal** [page 28] can be saved:

```
(define (equal a b)  
  (cond ((or (atom a) (atom b))  
        (eq a b))  
        (t (and (equal (car a) (car b))  
                  (equal (cdr a) (cdr b))))))
```

2.4 structural recursion

Here is the *replace* function:

```
(define (replace old new form)  
  (cond ((equal old form) new)  
        ((atom form) form)  
        (t (cons (replace old new (car form))  
                  (replace old new (cdr form))))))
```

Replace replaces each occurrence of *old* in *form* with *new*:

```
(replace 'b 'x '#aabbcc) => '#aaxxcc  
(replace 'old 'new '(old (old) old)) => '(new (new) new)
```

It can substitute data of any complexity in data of any complexity:

```
(replace '(g x) '(h x) '(f (g x))) => '(f (h x))
```

To do so, it uses a kind of recursion that is even more expensive than linear recursion.

Remember linear recursion? It is what happens when a function has to wait for a recursive call to complete.

Append2 (which first occurred on page 22) is linear recursive, because **cons** must wait until the recursive call to *append2* returns:

```
(define (append2 a b)  
  (cond ((null a) b)  
        (t (cons (car a)  
                  (append2 (cdr a) b)))))
```

In *replace*, **cons** must wait for the completion of *two* recursive calls.

Therefore, the space required for intermediate results of *replace* grows even faster than the space required by linear recursive functions:

```
(replace 'b 'x '((a . b) . (c . d)))  
-> (cons (replace 'b 'x '(a . b))  
        (replace 'b 'x '(c . d)))  
-> (cons (cons (replace 'b 'x 'a)  
              (replace 'b 'x 'b))  
        (cons (replace 'b 'x 'c)  
              (replace 'b 'x 'd)))
```

In the worst case *replace* adds two applications of itself each time it recurses. It has to do so, because all atoms of a pair have to be visited before *replace* can return. In this case, the space required by the function grows exponentially.

Because this kind of recursion is required to process recursive structures, it is called *structural recursion*.

When the structure to be traversed is “flat” — like a list — structural recursion is no more expensive than linear recursion:

```
(replace 'c 'x '#abc)  
-> (cons (replace 'c 'x 'a)  
        (replace 'c 'x '#bc))  
-> (cons (replace 'c 'x 'a)  
        (cons (replace 'c 'x 'b)  
              (replace 'c 'x '#c)))
```

zenstyle programming

```
-> (cons (replace 'c 'x 'a)
        (cons (replace 'c 'x 'b)
              (cons (replace 'c 'x 'c) ())))
```

Is there anything that can be done about structural recursion?

In the case of *replace*: no.

As the name already suggests:

Structural recursion is needed to traverse recursive structures.

There are not many occasions that require structural recursion, though.

The *contains* function traverses a recursive structure, but without combining the results of its recursive calls:

```
(define (contains x y)
  (cond ((equal x y) :t)
        ((atom y) :f)
        (t (or (contains x (car y))
                (contains x (cdr y))))))
```

Contains is similar to *replace*: its trivial cases return atoms, and its general case recurses twice.

However, the second recursive call to *contains* is a tail call.

Why?

When the first recursive call to *contains* returns truth, **or** does not even perform the second call and returns truth immediately.

When the first recursive call returns falsity, **:f** can be substituted for

```
(contains x (car y))
```

in

```
(or (contains x (car y))
    (contains x (cdr y)))
```

which leads to

```
(or :f (contains x (cdr y)))
```

and because

```
(or :f x) = (or x) = x
```

the second call to *contains* must *always* be a tail call.

The **or** pseudo function only waits for the first recursive call, which makes *contains* effectively

linear recursive.

The first recursive call cannot be eliminated, because it reflects an inherent property of the structure to be traversed.

For the same reason, **equal** [page 30] is linear recursive.

2.5 functions revisited

This is an *anonymous function*:

```
(lambda (topping) (list 'pizza 'with topping))
```

It is equivalent to the named function *pizza* [page 11]:

```
((lambda (topping) (list 'pizza 'with topping))  
 'pineapple)  
=> '(pizza with pineapple)
```

Anonymous functions are created by the pseudo function **lambda**:

```
(lambda (topping) (list 'pizza 'with topping))  
=> {closure (topping)}
```

Lambda creates a *closure* from the anonymous function.

Forms delimited by curly braces are *unreadable*:

```
{no matter what} => bottom
```

They are used to represent data that have no unambiguous external representation.

All zenlisp functions are either *primitive functions* or closures.

Define, **cdr**, and **lambda** itself are primitive functions:

```
define => {internal define}  
  cdr => {internal cdr}  
lambda => {internal lambda}
```

Other pre-defined zenlisp functions are closures:

```
reverse => {closure #a}  
  list => {closure x}
```

The code of **list** has been shown before:

```
(define (list . x) x)
```

Here is an anonymous function that implements **list**:

```
(lambda x x)
```


zen style programming

When the argument list of a lambda function is a single variable, that variable binds a list containing all actual arguments.

So lambda functions can be variadic, just like named functions.

To implement a variadic function that expects some mandatory arguments, dotted argument lists are used:

```
((lambda (x . y) y)) => bottom
((lambda (x . y) y) 'a) => ()
((lambda (x . y) y) 'a 'b) => '#b
((lambda (x . y) y) 'a 'b 'c) => '#bc
```

Lambda functions are totally equivalent to named functions.

In fact, they are one and the same concept.

```
(define (pizza top) (list 'pizza 'with top))
```

is just a short form of writing

```
(define pizza (lambda (top) (list 'pizza 'with top)))
```

A named function is nothing but a variable bound to a lambda function.

2.5.1 bound and free variables

Here is a definition:

```
(define (f x) (cons x :f))
```

A variable that occurs in the argument list of a function is said to be *bound in that function*.

The variable *x* is bound in the function

```
(lambda (x) (cons x :f))
```

but the variable *cons* is not bound inside of that function.

A variable that does not occur in the argument list of a function but *does* occur in the term of the same function is said to be *free in that function*.

F is free in

```
(lambda (x) (f x))
```

A variable is said to be bound in a function because the variable gets bound to a value when the function is called.

But what values do free variables get?

It depends on the *lexical context* of the function.

The (lexical) context of

```
(lambda (x) (f x))
```

is the *global* context. The global context is the context in which **define** binds its values.

In the global context, *f* was bound to the function

```
(lambda (x) (cons x :f))
```

by the **define** at the beginning of this section. Therefore *f* is bound to the same function inside of

```
(lambda (x) (f x))
```

BTW: the variable *cons* in the function

```
(define (f x) (cons x :f))
```

is bound to the primitive **cons** function. The primitive functions of zenlisp are defined by the system itself when it starts. They are also defined in the global context.

2.6 local contexts

Here is an expression with a *local context*:

```
(let ((topping-1 'extra-cheese)
      (topping-2 'pepperoni))
  (list 'pizza 'with topping-1 'and topping-2))
```

The variables *topping-1* and *topping-2* are created locally inside of **let**.

Let is a pseudo function of two arguments.

The first argument is called its *environment* and the second one its *body* (or *term*).

The environment of **let** is a list of two-member lists:

```
((name1 value1)
 ...
 (namen valuen))
```

Let evaluates all values and then binds each value to the name associated with that value.

The term is evaluated inside of the context created by binding the values to names locally, and the normal form of the term is returned:

```
(let ((x 'heads) (y 'tails))
  (cons x y))
=> '(heads . tails)
```

zen style programming

Here is a **let** whose body is another **let**:

```
(let ((x 'outer))
  (let ((x 'inner)
        (y x))
    (list x y)))
=> '(inner outer)
```

There are two local contexts in this expression, an *inner* one and an *outer* one.

A context that is *created by* a **let** is called its *inner context*.

The context in which **let** occurs is called its *outer context*.

Let evaluates all values of its environment in its *outer* context:

```
(let ((xouter 'outer))
  (let ((xinner 'inner)
        (y xouter))
    (list x y)))
=> '(inner outer)
```

In case of a duplicate symbol, the term of the inner **let** can only refer to the inner instance of that symbol, and the environment of the inner **let** can only refer to its outer instance.

BTW, **let** is nothing but an alternative syntax for the application of an anonymous function:

```
(let ((hd 'heads)      = ((lambda (hd tl)
    (tl 'tails))        (cons hd tl))
    (cons hd tl))      'heads
    'tails)
```

This equivalence explains neatly why values in environments are evaluated in outer contexts.

You would not expect a lambda function to evaluate its arguments in its inner context, would you?

2.6.1 closures

In nested local contexts, inner values shadow outer values:

```
(let ((v 'outer))
  (let ((v 'inner))
    v))
=> 'inner
```

But what happens when a local variable occurs as a free variable in a lambda function?

```
(let ((v 'outer))
  (let ((f (lambda () v)))
    (let ((v 'inner))
      (f))))
=> 'outer
```

What you observe here is a phenomenon called *lexical scoping*.

The value of the free variable v inside of the function f depends on the lexical context in which the function is *defined*.

The *definition* of f takes place in a context where v is bound to `'outer`, and the function f *memorizes* this binding.

The re-definition of v has no effect on it.

A function that memorizes the lexical values of its free variables is called a *closure*.

Here is a function that creates a closure:

```
(define (create-conser x)
  (lambda (y) (cons x y)))
```

The closure returned by the function conses x to its argument.

The value of x is specified when the closure is created:

```
(define cons-cherry (create-conser 'cherry))
(define cons-lemon (create-conser 'lemon))
```

The closures memorize the parameters passed to *create-conser*:

```
(cons-cherry 'pie) => '(cherry . pie)
(cons-lemon 'juice) => '(lemon . juice)
```

Alright, once again in slow motion:

```
((lambda (x) (lambda (y) (cons x y))) 'lemon)
→ (lambda (y) (cons 'lemon y))
```

This is a step called *beta reduction*. It is not a reduction in the sense of `zenlisp`, but a more abstract concept, as indicated by the generic arrow above.

Beta reduction replaces each variable of a lambda function which is free in the term of that function with the corresponding actual argument:

```
((lambda (y) (cons 'lemon y)) 'juice) → (cons 'lemon 'juice)
```

Y is free in **(cons 'lemon y)**, so it is replaced by the value associated with x .

Things are different here:

```
((lambda (x) (lambda (x) x)) 'foo) → (lambda (x) x)
```

Because x is bound in **(lambda (x) x)**, it is not replaced with `'foo`.

This is a mixed scenario:

```
((lambda (x) (list x (lambda (x) x))) 'foo) → (list 'foo (lambda (x) x))
```

zen style programming

The first x is free and therefore replaced; the second one is bound in a function and hence left alone.

Beta reduction is a transformation of the *lambda calculus* (LC). LC is the mathematical foundation of all LISP languages.

Beta reduction is a formal model for closures and parameter binding in `zenlisp` function calls.

2.6.2 recursive functions

Here is a simple function:

```
(define (d x)
  (or (atom x) (d (cdr x))))
```

Whatever you pass to d , it always returns `:t`:

```
(d 'x) => :t
(d '#xyz)
-> (d '#yz)
-> (d '#z)
-> (d ())
=> :t
```

Here is a not so simple question:

If **lambda** closes over all of its free variables and

```
(define (d x) (or (atom x) (d (cdr x))))
```

is equivalent to

```
(define d (lambda (x) (or (atom x) (d (cdr x)))))
```

then **lambda** must close over d before d gets bound to the resulting closure.

Then how can d recurse?

Obviously, it can.

The answer is that d is not really a closure at all (although it looks like one).

Whenever **define** binds a name to a lambda function, it stops **lambda** from capturing its free variables.

The result is a closure with an *empty environment*.

Because the function that is bound to d does not have a local value for d , it resorts to the global binding of d .

This method is called *dynamic scoping*.

It is called “dynamic”, because it allows to change the values of free variables dynamically.

Here is dynamic scoping in action:

```
(define food 'marmelade)
(define (get-food) food)
(get-food) => 'marmelade
(define food 'piece-of-cake)
(get-food) => 'piece-of-cake
```

Because dynamic scoping allows to bind values to symbols *after* using the symbols in functions, it can even be used to create *mutually recursive* functions:

```
(define (d1 x) (or (atom x) (d2 (cdr x))))
(define (d2 x) (or (atom x) (d1 (cdr x))))
(d1 '#xyz) => :t
```

Two functions *f* and *g* are mutually recursive if *f* calls *g* and *g* calls *f*.

BTW, you can use **let** with an empty environment to make **define** create a closure:

```
(define food 'marmelade)
(define get-food (let () (lambda () food)))
(get-food) => 'marmelade
(define food 'piece-of-cake)
(get-food) => 'marmelade
```

However, this method renders recursive functions impossible:

```
(define dc (let () (lambda (x)
                     (or (atom x)
                         (dc (cdr x))))))
(dc '#xyz) => bottom
```

2.6.3 recursive closures

Let can bind trivial functions:

```
(let ((cons-lemon
      (lambda (x)
        (cons 'lemon x))))
  (cons-lemon 'cake))
=> '(lemon . cake)
```

Let can *not* bind recursive functions, because the function closes over its own name *before* it is bound to that name:

```
(let ((d (lambda (x)
           (or (atom x)
               (d (cdr x))))))
  (d '#xyz))
=> bottom
```

zen style programming

Letrec can bind *recursive functions*:

```
(letrec ((d (lambda (x)
              (or (atom x)
                  (d (cdr x))))))
  (d '#xyz))
=> :t
```

This is why it is called `letrec`.

Using **letrec**, functions like **reverse** [page 21] can be packaged in one single **define**:

```
(define (reverse a)
  (letrec
    ((reverse2
      (lambda (a r)
        (cond ((null a) r)
              (t (reverse2 (cdr a)
                           (cons (car a) r))))))
    (reverse2 a ())))
```

Even *mutual recursion* can be implemented using **letrec**:

```
(letrec ((d1 (lambda (x)
                (or (atom x) (d2 (cdr x))))))
  (d2 (lambda (x)
        (or (atom x) (d1 (cdr x))))))
  (d1 '#xyz))
=> :t
```

The only difference between **let** and **letrec** is that **letrec** fixes recursive references in its environment after binding values to symbols.

Therefore

Local functions should be bound by <code>letrec</code>, local data by <code>let</code>.
--

2.6.4 recursive bindings

Here is an expression reducing to a closure:

```
(lambda () f)
```

Applications of the closure reduce to no value, because *f* has no value:

```
((lambda () f)) => bottom ; make sure that F is unbound!
```

Zenlisp has a set of *meta functions* that modify the state of the interpreter. The meta function `ap-
plication`

```
(closure-form env)
```

makes the interpreter display the lexical environments of closures when printing their normal forms:

```
(lambda () f) => (closure () f ((f . {void})))
```

The closure prints in round brackets because its full representation is unambiguous.

All four parts of the closure print now: the **closure** keyword, the argument list, the body, and the lexical environment captured by **lambda**.

The environment is stored in an *association list* (a.k.a. *alist*).

An association list is a list of pairs, where each car part of a pair is a key and each cdr part is a value *associated* with that key:

```
((key1 . value1) ... (keyn . valuen))
```

In lexical environments, variable names are keys and the values associated with the keys are the values of those variables.

The **assq** and **assoc** functions are used to retrieve associations from alists:

```
(define alist '((food . orange) (drink . milk)))  
(assq 'drink alist) => '(drink . milk)  
(assq 'soap alist) => :f
```

Assoc is related to **assq** in the same way as **member** is related to **memq** [page 15]:

```
(assq '(key) '(((key) . value))) => :f  
(assoc '(key) '(((key) . value))) => '((key) . value)
```

The problem with the alist of the closure

```
(closure () f ((f . {void})))
```

is that *f* is associated with no specific value — even if the closure itself is associated with the symbol *f* (the form **{void}** indicates that *f* is not bound at all):

```
(let ((f (lambda () f))) f)  
=> (closure () f ((f . {void})))
```

We mean to refer to the inner *f*, but **lambda** closes over the outer *f*.

Letrec uses a function called **recursive-bind** to fix environments containing recursive references:

```
(recursive-bind '((f . (closure () f ((f . wrong))))))
```

It does not matter what *f* is bound to in the inner context, because that is exactly what **recursive-bind** will change.

In the resulting structure, *wrong* is replaced with a reference to the value of the outer *f*:

zen style programming

```
(recursive-bind '((fouter . (closure () finner ((finner . wrong))))))  
=> '((fouter . (closure () finner ((finner . fouter)))))
```

Because f now contains a reference to f , it is a cyclic structure.

When you reduce above application of **recursive-bind** with **closure-form** set to `env`, the interpreter will attempt to print an infinite structure:

```
(recursive-bind '((f . (closure () f ((f . wrong))))))  
=> '((f . (closure () f  
      ((f . (closure () f  
            ((f . (closure () f  
                  ((f . ...
```

This is why only the argument lists of closures print by default. You can restore this behaviour by applying

```
(closure-form args)
```

2.7 higher-order functions

Here is a *higher-order function*:

```
(define (compose f g)  
  (lambda x (f (apply g x))))
```

A higher-order function is a function that takes a function as an argument and/or reduces to a function.

The *create-conser* function [page 37] introduced earlier also was a higher-order function.

The *compose* function takes two functions f and g as arguments and creates an anonymous function that implements the composition of f and g .

The **cadr** function, which extracts the second element of a list, can be implemented using *compose*:

```
(compose car cdr) => {closure #x}  
((compose car cdr) '(pizza with pepperoni)) => 'with
```

Of course, you would probably write **cadr** this way:

```
(define (cadr x) (car (cdr x)))
```

So here comes a more interesting example.

Filter extracts members with a given property from a flat list. The property is tested by applying the predicate p to each member:³

³ Writing a tail-recursive version of *filter* is left as an exercise to the reader.

```
(define (filter p lst)
  (cond ((null lst) ())
        ((p (car lst))
         (cons (car lst)
               (filter p (cdr lst))))
        (t (filter p (cdr lst)))))
```

Filter is a higher-order function because it expects a function — a predicate to be precise — in the place of *p*. It extracts members of a list that satisfy that predicate:

```
(filter atom '(orange '#xyz juice (a . b))) => '(orange juice)
```

A negative filter that removes all elements with a given property can be implemented on top of *filter* using a closure:

```
(define (remove p lst)
  (filter (lambda (x) (not (p x)))
          lst))
(remove atom '(orange #xyz juice (a . b))) => '(#xyz (a . b))
```

Not implements the logical “not”:

```
(define (not x) (eq x :f))
```

The function

```
(lambda (x) (not (p x)))
```

is in fact a useful higher-order function on its own. Here is a slightly improved version:

```
(define (complement p)
  (lambda x (not (apply p x))))
```

It turns a predicate into its complement:

```
(atom 'x) => :t
((complement atom) 'x) => :f
(eq 'apple 'orange) => :f
((complement eq) 'apple 'orange) => :t
```

2.7.1 mapping

Map maps functions over lists:

```
(map (lambda (x) (cons 'lemon x))
     '(cake juice fruit))
=> '((lemon . cake) (lemon . juice) (lemon . fruit))
```

Map implements list manipulation and flow control at the same time.

Here is a function that uses **map** to compute the depth of a list: ⁴

zen style programming

```
(require '~nmath) ; load natural math package
(define (depth x)
  (cond ((atom x) '#0)
        (t (+ '#1 (apply max (map depth x))))))
```

Map can map functions over any number of lists:

```
(map car '(#ab #cd #ef)) => '#ace

(map cons '#ace '#bdf) => '((a . b) (c . d) (e . f))

(map (lambda (x y z) (append x '#- y '#- z))
     '(#lemon      #cherry)
     '(#chocolate #banana)
     '(#ice-cream  #shake))
=> '(#lemon-chocolate-ice-cream
    #cherry-banana-shake)
```

Map-car is a simplified version of **map** that accepts only unary functions (functions of one variable) and only one single list:

```
(define (map-car f a)
  (letrec
    ((map-car2
      (lambda (a r)
        (cond ((null a) (reverse r))
              (t (map-car2 (cdr a)
                           (cons (f (car a)) r))))))
    (map-car2 a ())))
```

The *any-null* predicate checks whether any of the data contained in a list is equal to **()**:

```
(define (any-null lst)
  (apply or (map-car null lst)))
```

Mapping **null** over a list gives a list of truth values:

```
(map-car null '(#x #y () #z)) => '(:f :f :t :f)
```

Applying **or** to the resulting list gives truth if at least one member of the list is not **:f**.

The *car-of* and *cdr-of* functions map a list of lists to a list of car and cdr parts respectively:

```
(define (car-of x) (map-car car x))
(define (cdr-of x) (map-car cdr x))
```

Any-null, *car-of*, and *cdr-of* are used to implement **map**:

```
(define (map f . a)
  (letrec
```

⁴ **+** computes the sum of a list, **max** extracts the maximum of a list. **'#0** is zenlisp's way to express the number 0.

```
(map2
  (lambda (a b)
    (cond ((any-null a) (reverse b))
          (t (map2 (cdr-of a)
                    (cons (apply f (car-of a)) b))))))
  (map2 a ())))
```

Because *any-null* checks whether *any* of the sublists of its argument is **()**, **map** returns as soon as the end of the shortest list passed to it is reached:

```
(map cons '#ab '#abcd) => '((a . a) (b . b))
```

2.7.2 folding

Fold folds lists by combining adjacent members:

```
(fold cons 'a '(b c d)) => '(((a . b) . c) . d)
```

The first argument of **fold** is the function used to combine elements, the second argument is the “base” element, and the third one is the list of elements to combine. The base element is combined with the first member of the list.

When the list of members is empty, **fold** returns the base element:

```
(fold cons 'empty ()) => 'empty
```

Fold is used to iterate over lists.

Predicate-iterator is a higher-order function that uses **fold** to make a two-argument predicate variadic ⁵ (**Neq** is **(compose not eq)**):

```
(define (predicate-iterator pred)
  (let ((:fail ':fail))
    (let ((compare
            (lambda (a b)
              (cond ((eq a :fail) :fail)
                    ((pred a b) b)
                    (t :fail)))))
      (lambda (first . rest)
        (neq (fold compare first rest) :fail)))))
```

Here is **predicate-iterator** in action:

```
(define eq* (predicate-iterator eq))
(eq*) => bottom
(eq* 'lemon) => :t
(eq* 'lemon 'lemon) => :t
(eq* 'lemon 'lemon 'lemon) => :t
(eq* 'lemon 'lemon 'orange 'lemon) => :f
```

⁵ The **predicate-iterator** function has a minor flaw. See page 55 for details.

zen style programming

Should an application of eq^* to less than two arguments really reduce to $:t$? If not, how would you fix **predicate-iterator**?

Fold reduces a list by combining its members *left-associatively*:

```
(fold f 'a '(b c d)) = (f (f (f a b) c) d)
```

An operation is left-associative, if multiple applications of the same operator *associate to the left*, i.e. bind stronger to the left.

The mathematical “minus” operator is left-associative, because

$$a - b - c - d = (((a - b) - c) - d)$$

To fold the members of a list right-associatively, the **fold-r** function is used:

```
(fold-r f 'a '(b c d)) = (f a (f b (f c d)))
```

An example for right-associative operators is mathematical exponentiation:

$$\frac{b^{c^d}}{a} = a^{(b^{(c^d)})}$$

The following reductions illustrate the difference between **fold** and **fold-r**.

```
(fold cons 'a '(b c d)) => '(((a . b) . c) . d)
(fold-r cons 'a '(b c d)) => '(b . (c . (d . a)))
                        = '(b c d . a)
```

3. rather esoteric aspects

3.1 numeric functions

Before a `zenlisp` program can work with numbers, it has to *require* one of the math packages:

```
(require '~nmath)
```

Nmath is a package containing functions for computations with *natural numbers* (plus zero).

Zenlisp uses lists of digits to represent natural numbers. Here are some examples:

```
'#0   '#3   '#1415  '#926535
```

Here is the **length** function:

```
(define (length x)
  (letrec
    ((length2
      (lambda (x r)
        (cond ((null x) r)
              (t (length2 (cdr x) (+ '#1 r))))))
    (length2 x '#0)))
```

Length computes the length of a list:

```
(length ()) => '#0
(length '(orange juice)) => '#2
(length '#4142135623) => '#10
```

It uses the math function **+**, which adds numbers:

```
(+) => '#0
(+ '#5) => '#5
(+ '#5 '#7) => '#12
(+ '#5 '#7 '#9) => '#21
```

Like many other math functions, **+** is *variadic*.

Passing zero arguments to **+** yields `'#0`, the neutral element of the mathematical “plus” operation.

Here are some other math functions:

```
(*) => '#1
(* '#5 '#7 '#9) => '#315
(- '#7 '#3 '#4) => '#0
```

The ***** function computes the product of its arguments and **-** computes their difference.

zen style programming

Integer division is performed by the **divide** function:

```
(divide '#17 '#3) => '(#5 #2)
```

It returns a list containing the integer quotient and the division remainder of its arguments:

```
(divide dividend divisor) => '(quotient remainder)
```

When only the quotient or the remainder of two numbers is of interest, the **quotient** and **remainder** functions are useful:

```
(quotient '#17 '#3) => '#5  
(remainder '#17 '#3) => '#2
```

The division remainder of two numbers *a* and *b* is defined as

(- a (* (quotient a b) b)).

(Expt x y) computes *x* to the power of *y*:

```
(expt '#3 '#100) => '#515377520732011331036461129765621272702107522001
```

Because **zenlisp** implements numbers as lists of digits, precision is only limited by time and memory.

This approach is known as *bignum arithmetics* (big number arithmetics). It guarantees that numeric expressions always yield a mathematically correct result (which includes, of course, the possibility of returning *bottom*).

3.1.1 numeric predicates

Here is the code of the *ngcd* function:

```
(define (ngcd a b)  
  (cond ((zero b) a)  
        ((zero a) b)  
        ((< a b) (ngcd a (remainder b a)))  
        (t (ngcd b (remainder a b)))))
```

Ngcd computes the *greatest common divisor* of two natural numbers:

```
(ngcd '#12 '#6) => '#6  
(ngcd '#289 '#34) => '#17  
(ngcd '#17 '#23) => '#1
```

Of course you would use the **gcd** function of one of the **zenlisp** math packages to compute the **gcd**, but *ngcd* uses two interesting functions.

The **zero** function tests whether its argument is equal to the number 0:

```
(zero '#0) => :t  
(zero '#1) => :f
```

The argument of **zero** must be a *number*:

```
(zero 'non-number) => bottom
```

Zero is a *numeric predicate*.

It can be expressed (albeit less efficiently) using another numeric predicate:

```
(define (zero x) (= x '#0))
```

The **=** predicate tests whether its arguments are numerically equal:

```
(= '#3 '#3 '#3) => :t  
(= '#23 '#17) => :f
```

Here are some other numeric predicates:

```
(< '#1 '#2 '#3) => :t  
(> '#3 '#2 '#1) => :t  
(<= '#1 '#2 '#2) => :t  
(>= '#2 '#1 '#1) => :t
```

They test for the properties “*less than*”, “*greater than*”, “*less than or equal to*”, and “*greater than or equal to*”, respectively.

The following relation holds for each comparative numeric predicate *R*:

$$(R\ a_1\ \dots\ a_n) = (\text{and}\ (R\ a_1\ a_2)\ \dots\ (R\ a_{n-1}\ a_n))$$

So, for example,

```
(< a b c d)
```

can be written as

```
(and (< a b) (< b c) (< c d))
```

This is exactly the reason why there is *no* negative equivalence operator.

If there was such an operator (let us call it *not=*), the expression

```
(not= a1 ... an)
```

would translate to

```
(and (not= a1 a2) ... (not= an-1 an))
```

while

```
(not (and (= a1 a2) ... (= an-1 an)))
```

equals

```
(or (not= a1 a2) ... (not= an-1 an))
```


zen style programming

3.1.2 integer functions

When using the **nmath** package, negative results cannot occur:

```
(- '#0 '#1) => bottom
```

Loading the **imath** (integer math) package extends the domain and/or range of the natural math functions to include negative numbers:

```
(require '~imath)
(- '#0 '#1) => '#-1
```

Loading **imath** loads **nmath** automatically.

After loading the integer math package, most functions discussed in the previous section accept negative arguments, too:

```
(+ '#5 '#-7) => '#-2
(* '#-5 '#-5) => '#25
(quotient '#17 '#-3) => '#-5
```

The **-** function is extended to handle single arguments, implementing the “negate” operator:

```
(- '#27182) => '#-27182
```

In addition to the **remainder** function **imath** introduces the **modulo** function. The modulus of a and b is defined as:⁶

$a - \text{floor}(a/b) \times b$

The results of **remainder** and **modulo** differ only when their arguments a and b have different signs:

argument a	argument b	remainder	modulo
'#+23	'#+5	'#+3	'#+3
'#+23	'#-5	'#+3	'#-2
'#-23	'#+5	'#-3	'#+2
'#-23	'#-5	'#-3	'#-3

Because **zenlisp** lacks a “floor” function, **modulo** exploits the fact that

```
(modulo x y) = (+ y (remainder x y))
```

if x and y have different signs.

Here is **modulo**:

```
(define (modulo a b)
  (let ((rem (remainder a b)))
```

⁶ The “floor” of x denotes the largest integer that is not larger than x .

```
(cond ((zero rem) '#0)
      ((eq (negative a) (negative b))
       rem)
      (t (+ b rem))))
```

The **negative** predicate tests whether its argument (which must be numeric) is negative.

Eq is used as a *logical equivalence* operator in **modulo**:

```
(eq (negative a) (negative b))
```

reduces to truth if either both *a* and *b* are positive or both are negative.

3.1.3 rational functions

When using the **imath** package, results *between* zero and one cannot occur:

```
(expt '#2 '#-5) => bottom
```

Loading the **rmath** (rational math) package extends the domain and/or range of the integer math functions to include rational numbers:

```
(require '~rmath)
(expt '#2 '#-5) => '#1/32
```

The format of rational numbers is

```
'#numerator/denominator
```

where *numerator* and *denominator* are integer numbers.

Negative rational numbers have a negative numerator:

```
(- '#1 '#3/2) => '#-1/2
```

The **numerator** and **denominator** functions extract the individual parts of a rational number:

```
(numerator '#-5/7) => '#-5
(denominator '#-5/7) => '#7
```

Even if the **rmath** package was loaded, the **divide** and **quotient** functions still perform integer division. The **/** function performs the division of rational numbers:

```
(/ '#10 '#2) => '#5
(/ '#20 '#6) => '#10/3
(/ '#1/2 '#1/2) => '#1
```

The **sqrt** function of the **nmath** package delivers the integer part of the square root of a number. **Rmath** extends this function to deliver a rational result:

```
(sqrt '#144) => '#12
(sqrt '#2) => '#665857/470832
```

zen style programming

The precision of the rational **sqrt** function is controlled by the ***epsilon*** variable:

```
*epsilon* => '#10
```

An ***epsilon*** value of 5, for example, denotes a precision of 10^{-5} , i.e. when converted to fixed point notation, the result of **sqrt** has a precision of (at least) 5 decimal places. The default precision is 10^{-10} .

3.1.4 type checking and conversion

The **number-p** predicate checks whether a datum represents a number:

```
(number-p '#314) => :t  
(number-p '#-159) => :t  
(number-p '#-265/358) => :t
```

Any type of datum can be passed to **number-p**:

```
(number-p 'marmelade) => :f  
(number-p '(heads . tails)) => :f
```

The **natural-p**, **integer-p**, and **rational-p** predicates test whether a number has a specific type:

```
(natural-p '#1) => :t  
(integer-p '#-1) => :t  
(rational-p '#1/2) => :t
```

A datum passed to one of these predicates *must be a valid number*:

```
(integer-p 'sushi) => bottom
```

Also note that these predicates only check the syntax of the data passed to them. Therefore

```
(natural-p '#+1) => :f  
(integer-p '#1/1) => :f  
(rational-p '#5) => :f
```

Although $\#1/1$ is an integer value, **integer-p** does not recognize it as such because it has the syntax of a rational number.

To check whether the *value* of a number has a specific type, the datum representing the number must be *normalized* first. This can be done by applying a neutral operation:

```
(natural-p (+ '#0 '#+1)) => :t  
(integer-p (* '#1 '#1/1)) => :t
```

However, this works only if the argument has the syntax of a “more complex” type than the one checked against:

```
(rational-p '(+ '#0 '#5)) => :f
```

Although `'#5` is a valid rational number, **rational-p** does not return truth because the normalized form of `'#5` is still `'#5` and not `'#5/1`.

This is what normalization does:

- reduce rationals to least terms;
- move the signs of rationals to the numerator;
- remove denominators of 1;
- remove plus signs from integers;
- remove leading zeroes.

Normalization reduces each number to its *least applicable type*.

Rationals are a superset of integers and integers are a superset of naturals.

The least type of a number is the type representing the smallest set that includes that number.

For instance, `'#-5/-5` is a rational number, but normalizing it yields a natural number:

```
(+ '#0 '#-5/-5)
-> '#-5/-5      ; added '#0
-> '#1/1        ; reduced to least terms
=> '#1          ; removed denominator of '#1
```

The **natural**, **integer**, and **rational** functions attempt to convert a number to a different type:

```
(natural '#+5) => '#5
(integer '#5)  => '#5
(rational '#-5) => '#-5/1
```

In case the requested conversion cannot be done, these functions reduce to bottom, thereby implementing numeric *type checking*:

```
(natural '#-1) => bottom
(integer '#1/2) => bottom
```

The **rmath** package is required for the following type conversion to work:

```
(require '~rmath)
(integer '#4/2) => '#2
```

It will not work when using just **imath**:

```
(require '~imath)
(integer '#4/2) => bottom
```

Type conversion functions are used to guard functions against arguments that are not in their domains.

Here is a higher-order function that turns a two-argument math function into a variadic math function:

zen style programming

```
(define (arithmetic-iterator conv fn neutral)
  (lambda x
    (cond ((null x) neutral)
          (t (fold (lambda (a b)
                     (fn (conv a) (conv b)))
                   (car x)
                   (cdr x))))))
```

The *fn* argument is the function to convert, *conv* is one of the type conversion functions, and *neutral* is a base value for **fold**.

Let *n+* be an unguarded binary function (a function of two variables) for adding natural numbers:

```
(n+ '#1 '#2) => '#3
(n+ '#+1 '#+2) => bottom
(n+ '#1 '#2 '#3) => bottom
```

Arithmetic-iterator turns this function into a guarded variadic function:

```
((arithmetic-iterator natural n+ '#0) '#+1 '#+2 '#+3) => '#6
((arithmetic-iterator natural n+ '#0) '#+0 '#-1) => bottom
```

Arithmetic-iterator is actually used to implement most of the zenlisp math functions.

3.2 side effects

The *effect* of a *function* is to map values to values:

```
(null ()) => :t
(null x) => :f ; for any x that does not equal ()
```

The effect of **null** is to map **()** to **:t** and any other value to **:f**.

Each time a function is applied to the same actual argument(s) it returns the same result:

```
(atom 'x) => :t
(atom 'x) => :t
(atom 'x) => :t
```

This is a fundamental property of a function.

An effect that cannot be explained by mapping values to values is called a *side effect*.

Define is known to have a side effect:

```
(define marmelade 'medium-cut-orange) => 'marmelade
marmelade => 'medium-cut-orange
(define marmelade 'fine-cut-orange) => 'marmelade
marmelade => 'fine-cut-orange
```

The mutation of the value of the variable *marmelade* cannot be explained by mapping *marmelade* and *'fine-cut-orange* to *'marmelade*.

Define does not even deliver the same value for each pair of arguments:⁷

```
(define define :f) => 'define  
(define define :f) => bottom
```

Because the first application of **define** changes the value of *define* itself to **:f**, the second form is not even a valid expression.

A function that has a side effect is a *pseudo function*, but not every pseudo function has a side effect. **Cond**, **quote**, **let**, and **lambda** are examples for pseudo functions without side effects.

3.2.1 subtle side effects

Neither **cons** nor **eq** has a side effect, but by bending the rules a little bit, they can be used to construct one.

Cons is guaranteed to deliver a *fresh* pair.

Eq compares data by comparing their locations in the memory of the computer. This is, of course, an implementation detail. You do not need to memorize it, but its implications are interesting:

```
(eq (cons x y) (cons x y)) => :f for any values of x and y.
```

This is the exception to the rule that passing two pairs to **eq** yields bottom:

Two applications of cons can never yield identical results.

Hence **cons** can create an absolutely *unique instance* of a datum:

```
(define unique-instance (cons 'unique-instance ()))
```

and **eq** can identify that instance:

```
(eq unique-instance unique-instance) => :t  
(eq unique-instance '(unique-instance)) => :f  
unique-instance => '(unique-instance)
```

This effect cannot be explained by mapping values to values, so it obviously is a side effect.

The creation of unique instances is the only reason why this side effect is described here. It can be used to overcome a subtle limitation of the **predicate-iterator** function [page 45].

Predicate-iterator uses the symbol **:fail** to represent failure. Hence it can deliver a wrong value when the function returned by it is applied to **:fail** itself:

```
((predicate-iterator equal) ' :fail ' :fail) => :f
```

⁷ If you actually try this, you will have to restart **zenlisp** to restore the original behaviour.

zen style programming

No matter what symbol you would invent in the place of `:fail`, the function created by **predicate-iterator** would never handle that symbol correctly.

What you need is a *unique instance* of the datum representing failure:

```
(define (predicate-iterator pred)
  (let ((:fail (cons ':fail ())))
    (let ((compare
           (lambda (a b)
             (cond ((eq a :fail) :fail)
                   ((pred a b) b)
                   (t :fail))))))
      (lambda (first . rest)
        (neq (fold compare first rest) :fail)))))
```

In the improved version of **predicate-iterator**, *:fail* binds to a unique instance of `'(:fail)`. Therefore, the function can even handle other instances of that datum correctly:

```
((predicate-iterator equal) '(:fail) '(:fail)) => :t
```

3.2.2 evaluation

The **eval** function interprets `zenlisp` expressions:

```
(eval '(letrec
  ((n (lambda (x)
        (cond ((atom x) ())
              (t (cons 'i (n (cdr x)))))))
    (n '(1 2 3 4 5))))
=> '#iiiiii
```

Note that the argument of **eval** is a *list* and therefore it is not evaluated before it is passed to **eval**. It is the **eval** function that gives meaning to a form, thereby turning it into an expression.

Eval reduces to bottom, if the datum passed to it does not constitute a valid expression:

```
(eval '(cons 'x)) => bottom
```

Eval has a side effect, if the expression interpreted by it has a side effect:

```
(eval '(define bar 'foo)) => 'bar
bar => 'foo
```

3.3 metaprogramming

Here is a trivial *metaprogram*:

```
(list 'lambda '(x) 'x) => '(lambda (x) x)
```

A metaprogram is a program that writes or rewrites a program.
--

Programs are expressions, and expressions are a subset of forms.

Every program can be turned into a datum by *quoting* it:

```
(quote (lambda (x) x)) => '(lambda (x) x)
```

Here is another simple meta program:

```
((lambda #x (list x (list 'quote x)))  
'(lambda #x (list x (list 'quote x))))
```

What is interesting about this kind of program is that its normal form is equal to its own code; *it reduces to itself*:⁸

```
((lambda #x (list x (list 'quote x)))  
'(lambda #x (list x (list 'quote x))))  
=> ((lambda #x (list x (list 'quote x)))  
    '(lambda #x (list x (list 'quote x))))
```

Such programs are widely known as *quines*. They are named after the philosopher W.V.O. Quine, who did a lot of research in the area of indirect self reference.

The above program inserts the quoted “half” of its code in the place of *x* in the form

```
(list x (list 'quote x))
```

thereby creating code that applies this half to a quoted copy of itself. The self reference is indirect because the expression does not reference itself, but *creates* a reference to itself.

Here is another classic self-referential expression:

```
((lambda #x #xx) (lambda #x #xx))
```

Can you deduce its normal form? Does it have one?

3.3.1 programs hacking programs

Here is a not so trivial metaprogram:

```
(define (let->lambda let-expr)  
  (let ((env (cadr let-expr)))  
    (let ((vars (map car env))  
          (args (map (lambda (x) (unlet (cadr x)))  
                      env))  
          (body (unlet (caddr let-expr))))  
      (append (list (list 'lambda vars body))  
              args))))
```

⁸ Argument lists are condensed in this example, because this is how the zenlisp printer prints them. This has no effect on the semantics, though. For example, `(lambda (x) (x x))` is equal to `(lambda #x #xx)`.

zen style programming

```
(define (unlet x)
  (cond ((atom x) x)
        ((eq (car x) 'quote) x)
        ((eq (car x) 'let) (let->lambda x))
        ((eq (car x) 'lambda)
         (list 'lambda
               (cadr x)
               (unlet (caddr x)))))
  (t (map unlet x))))
```

Let->lambda converts (a datum representing) a **let** expression to (a datum representing) the application of a lambda function:

```
(let->lambda '(let ((x 'heads)
                   (y 'tails))
                (cons x y)))
=> '((lambda #xy
      (cons x y))
   'heads 'tails)
```

The *unlet* function traverses (a datum representing) a zenlisp expression and replaces each **let** contained in it:

```
(unlet '(let ((y (let ((a 'orange)))
                a))
          (z '(cookies)))
      (let ((x '(i like)))
        (append x y z))))
=> '((lambda #yz
      ((lambda #x
        (append x y z))
       '(i like)))
   ((lambda #a a)
    'orange))
   '(cookies))
```

If nothing else, this program demonstrates that **let** is in some situations much more readable than **lambda**.

Unlet recurses through **map**. Recursion through map is always a *structural recursion*.

BTW, the default case (**map unlet x**) *cannot be replaced with*

```
(cons (unlet (car x))
      (unlet (cdr x)))
```

because doing so would break the test for quotation. Given

```
x = '(list quote x)
```

the above solution would lead to

```
(cons (unlet 'list)
      (unlet '(quote x)))
```

which in turn would result in *unlet* treating *x* as a quoted symbol, which it is not.

3.3.2 beta reduction by substitution

This section introduces a slightly more elaborate metaprogram.

Lambda-rename renames the variables of lambda expressions in such a way that each variable gets a unique name:

```
(lambda-rename '(lambda (x) (lambda (y) (cons x y))))
=> '(lambda (x:0) (lambda (y:1) (cons x:0 y:1)))
```

Its purpose is to resolve conflicts between variables:

```
(lambda-rename '(lambda (x) (list x (lambda (x) x))))
=> '(lambda (x:0) (list x:0 (lambda (x:1) x:1)))
```

Lambda-rename uses a few helper functions.

The *add* function adds a colon and a number to a symbol name:

```
(require '~nmath) ; this will be needed later
(define (add name level)
  (implode (append (explode name)
                    '#:
                    level)))
```

It uses the **implode** function to create a *new symbol name* from a list of single-character names:

```
(implode '(n e w - n a m e)) => 'new-name
```

Zenlisp numbers already *are* lists of symbols:

```
(cons 'digits: '#31415) => '(digits: 3 1 4 1 5)
```

In order to append '#:' and a number to an existing symbol, the existing symbol must be exploded first.

The **explode** function implements the reverse operation of **implode**:⁹

```
(explode 'symbol) => '(s y m b o l)
```

Lambda-rename keeps substitutions of old and new names in an association list, e.g:

```
'((y . y:1) (x . x:0))
```

Ext-sub takes a list of variable names, an alist, and a number. It adds a new substitution for each

⁹ Zenlisp will print '(s y m b o l) as '#symbol.

zen style programming

variable name passed to it:

```
(ext-sub '((x . x:2)) '(a b c) '#3)
=> '((c . c:3) (b . b:3) (a . a:3) (x . x:2))
```

Here is *ext-sub*:

```
(define (ext-sub sub vars level)
  (letrec
    ((add-var
      (lambda (name alist)
        (cons (cons name (add name level))
              alist))))
    (cond ((null vars) sub)
          ((atom vars) (add-var vars sub))
          (t (ext-sub (add-var (car vars) sub)
                      (cdr vars)
                      level)))))
```

Ext-sub handles variadic argument lists properly:

```
(ext-sub '() '(x . y) '#1) => '((y . y:1) (x . x:1))
```

The *subst* function is similar to **assoc**:

```
(define (subst name sub)
  (let ((v (assq name sub)))
    (cond (v (cdr v))
          (t name))))
```

Its only difference to **assoc** is that it returns the value in case of success and the key if no matching association is found:

```
(subst 'x '((x . x:0))) => 'x:0
(subst 'cons '((x . x:0))) => 'cons
```

Rename-vars traverses a datum representing an expression and renames all variables of lambda functions in both their argument lists and bodies:

```
(define (rename-vars expr sub level)
  (cond ((atom expr) (subst expr sub))
        ((eq (car expr) 'quote) expr)
        ((eq (car expr) 'lambda)
         (let ((vars (cadr expr))
               (body (caddr expr)))
           (let ((new-sub (ext-sub sub vars level)))
             (list 'lambda
                   (rename-vars vars new-sub level)
                   (rename-vars body
                               new-sub
                               (+ '#1 level))))))
        (t (map-car-i (lambda (x)
                        (rename-vars x sub level))
                      expr))))
```

Because *rename-vars* has to be able to process variadic argument lists of **lambda**, it cannot recurse through **map**. It uses a special version of *map-car* [page 44] that is able to process dotted (improper) lists (hence the trailing "i" in its name):

```
(define (map-car-i f a)
  (letrec
    ((map-car-i2
      (lambda (a r)
        (cond ((null a) (reverse r))
              ((atom a) (append (reverse r) (f a)))
              (t (map-car-i2 (cdr a)
                             (cons (f (car a)) r))))))
    (map-car-i2 a ())))
```

Two of the arguments of *rename-vars* are constant, so here is a wrapper that provides them:

```
(define (lambda-rename expr)
  (rename-vars expr () '#0))
```

After renaming the variables of lambda expressions using *lambda-rename*, *beta reduction* is nothing more than a simple substitution.

The *subst-vars* function substitutes each symbol of a given expression with the value associated with that symbol:

```
(define (subst-vars expr sub)
  (cond ((atom expr) (subst expr sub))
        ((eq (car expr) 'quote) expr)
        (t (map-car-i (lambda (x)
                        (subst-vars x sub))
                       expr))))
```

Using *lambda-rename* and *subst-vars*, beta reduction can be done without using closures:

```
(define (beta-reduce app)
  (let ((app (lambda-rename app)))
    (let ((vars (cadar app))
          (args (cdr app))
          (body (caddar app)))
      (subst-vars body (map cons vars args)))))
```

Instead of creating closures, *beta-reduce* substitutes free variables with their values:

```
(beta-reduce '((lambda (x) x) v))
=> 'v
(beta-reduce '((lambda (f) (lambda (x) (f x))) not))
=> '(lambda (x:1) (not x:1))
(beta-reduce '((lambda (x) (list x (lambda (x) x))) v))
=> '(list v (lambda (x:1) x:1))
```

You can use **eval** to interpret the output of metaprograms:

zen style programming

```
(eval (beta-reduce '((lambda (x) (list x (lambda (x) x))) 'v)))  
=> '(v {closure (x:1) x:1})
```

3.4 packages

This is a *zenlisp package*:

```
(define square :t)           ; name the package  
(require '~rmath)           ; declare dependencies  
(define (square x) (* x x)) ; package body
```

The **require** function loads a package with the given name only if the name is not bound. For instance

```
(require '~rmath)
```

loads **rmath** only, if the symbol **rmath** is not bound to any value. If the symbol is bound, **require** assumes that the package already has been loaded.

The leading tilde (“~”) makes *zenlisp* load a package from a standard location.

Because **require** loads packages only once, recursive references pose no problem. Loading a file named `foo.l` containing the package

```
(define foo :t)  
(require 'foo) ; require myself
```

will just load `foo`. No infinite recursion will occur.

part two

algorithms

4. list functions

4.1 heads and tails

Just for warming up, how do you find out whether a list x is the head of a list y , e.g.:

```
(headp '#a '#abc) => :t
(headp '#ab '#abc) => :t
(headp '#abc '#abc) => :t
(headp '#abcd '#abc) => :f
```

A list x is the head of a list y , if their leading members up to the length of x are equal. Here is *headp*, which performs this test:

```
(define (headp x y)
  (cond ((null y) (null x))
        ((null x) :t)
        (t (and (equal (car x) (car y))
                  (headp (cdr x) (cdr y))))))
```

According to *headp*, the empty list is the head of any list:

```
(headp () '(foo bar baz)) => :t
```

Do you think this is the right way to handle the empty “head”? If so, why? If not, why not? How does the fact that the car part of a list is also called the “head” of a list contribute to this controversy? (Q1)

While we are at it: how would you check whether a list x is the tail of a list y . Do not ponder too long:

```
(require 'headp)
(define (tailp x y)
  (headp (reverse x) (reverse y)))
```

Why is that fact that

```
(tailp () '#whatever) => :t
```

less controversial than the fact that

```
(headp () '#whatever) => :t
```

Bonus exercise: find better names for *headp* and *tailp*.

zen style programming

4.2 find the *n*'th tail of a list

The *nth* function extracts the tail of a list that starts at the *n*'th element of that list. When the list is too short, it returns **:f**:

```
(nth '#0 '#abc) => '#abc
(nth '#1 '#abc) => '#bc
(nth '#2 '#abc) => '#c
(nth '#5 '#abc) => :f
```

Here is the code of *nth*:

```
(require '~nmath)

(define (nth n x)
  (cond ((zero n) x)
        ((null x) :f)
        (t (nth (- n '#1)
                  (cdr x)))))
```

When the argument *n* is the same as the the length of the list, *nth* returns an empty list. In particular:

```
(nth '#0 ()) => ()
```

When you swap the first two predicates of the **cond** of *nth*, it will return **:f** in the above case. Which version do you consider more consistent? Why?

Do you think it is a good idea that the first list member has an offset of **'#0**? Implement a version that starts numbering members at **'#1**. What are the advantages and disadvantages of each version? What will your version return when you pass an offset of zero to it?

4.3 count the atoms of a form

The *count* function counts the atoms of a form recursively:

```
(count ()) => '#0
(count 'a) => '#1
(count '(a (b (c) d) e)) => '#5
```

Here is the code of the *count* function:

```
(require '~nmath)

(define (count x)
  (cond ((null x) '#0)
        ((atom x) '#1)
        (t (+ (count (car x))
                (count (cdr x))))))
```

Strictly speaking, *count* counts only the symbols of a form, because **()** is also an atom. If *count*

returned `'#1` for `()`, though, its results would be counter-intuitive. Why? (Q2)

Count uses structural recursion. Do you think it can be written in a more efficient way?

This is an interesting question. Here is a version of *count* that uses tail calls exclusively:

```
(require '~nmath)

(define (count1 x)
  (letrec
    ((c (lambda (x r s)
          (cond ((null x)
                 (cond ((null s) r)
                       (t (c (car s) r (cdr s))))))
      ((atom x)
       (c () (+ '#1 r) s))
      (t (c (car x) r (cons (cdr x) s))))))
    (c x '#0 ())))
```

It uses the internal function *c* that keeps its intermediate result (the atoms counted so far) in the extra parameter *r*. This technique was used earlier in this book to convert linear recursive programs to tail-recursive ones. Can this technique also be applied to turn structural recursion into tail recursion?

The *c* function uses an additional parameter *s* which stores the nodes to be visited in the future. Whenever it finds a new pair, it saves its *cdr* part in *s* and then starts traversing the *car* part of that pair.

When *count* finds an instance of `()`, it removes the head of *s* and traverses the structure that was stored there. Of course, the name *s* should invoke the connotation of a “stack”, and this is exactly what it is.

The only difference between the structural recursive version and the version using tail calls is that the version using tail calls stores activation records on an explicit stack rather than on *zenlisp*’s internal stack. The recursion itself cannot be removed. It is inherent in the problem.

This particular case illustrates nicely that attempting to remove recursion from solutions for inherently recursive problems often decreases readability while it barely increases efficiency.

4.4 flatten a tree

A tree is “flattened” by turning it into a flat list that contains the same member in the same order as the original tree:

```
(flatten '((a) (b (c)) (d (e (f))))) => '#abcdef
```

Here is the code of *flatten1*, which performs this operation — although in a highly inefficient way:

zen style programming

```
(define (flatten1 x)
  (cond ((null x) x)
        ((atom x) (list x))
        (t (append (flatten1 (car x))
                     (flatten1 (cdr x))))))
```

Can you name a few reasons why this implementation is far from efficient? (Yes, the obvious structural recursion is only one of them.)

Here is an improved version. It uses a clever trick to avoid structural recursion:

```
(define (flatten x)
  (letrec
    ((f (lambda (x r)
          (cond ((null x) r)
                ((atom x) (cons x r))
                (t (f (car x)
                      (f (cdr x) r))))))
    (f x ())))
```

The function still applies itself twice, but the result of the first application is passed to the second one, which is a tail call, thereby turning structural recursion into linear recursion.

Can this version of *flatten* be transformed into a version using tail calls, like *count*? Would this transformation improve efficiency any further? (Q3)

4.5 partition a list

In the previous part the functions *filter* and *remove* were introduced [page 42]. One of the functions extracts members satisfying a predicate, the other one removes members satisfying a predicate.

Can you write a program that combines the functions of *filter* and *remove*? Hint: the function should return two lists.

Here is the code of *partition*, which partitions a list into members satisfying and not satisfying a given predicate:

```
(define (partition p a)
  (letrec
    ((partition3
      (lambda (a r+ r-)
        (cond ((null a)
              (list r+ r-))
              ((p (car a))
               (partition3 (cdr a)
                           (cons (car a) r+)
                           r-))
              (t (partition3 (cdr a)
                              r+
                              (cons (car a) r-))))))
    (partition3 (reverse a) () ())))
```

The first member of its result is equal to the output of *filter* and its second member resembles the output of *remove*. Would it make sense to implement *filter* and *remove* on top of *partition*?

4.6 folding over multiple lists

The fold function (explained on page 45, code on page 271) folds lists to values:

```
(fold (lambda (x y) (list 'op x y)) '0 '(a b c))  
=> '(op (op (op 0 a) b) c)
```

The Revised⁶ Report on the Algorithmic Language Scheme (R⁶RS) — although being highly controversial — defines a few interesting functions. One is *fold-left*, which allows to fold over multiple lists:

```
(fold-left (lambda (x y z) (list 'op x y z))  
           '0  
           '(a b c)  
           '(d e f))  
=> '(op (op (op 0 a d) b e) c f)
```

All lists must have the same length. Can you implement *fold-left*? Hint: this function has some things in common with **map** [page 44] and **fold**. In particular, the following functions are helpful when implementing it:

```
(define (car-of a) (map car a))  
(define (cdr-of a) (map cdr a))
```

In fact, *fold-left* is exactly like **fold**, but uses the techniques of **map** to handle variadic arguments:

```
(define (fold-left f b . a*)  
  (letrec  
    ((fold  
      (lambda (a* r)  
        (cond ((null (car a*)) r)  
              (t (fold (cdr-of a*)  
                      (apply f r (car-of a*)))))  
        (cond ((null a*) (bottom 'too-few-arguments))  
              (t (fold a* b))))))  
    (fold a* b)))
```

The R⁶RS also defines a variant of **fold-r** [page 271] that can handle multiple lists. Unsurprisingly it is called *fold-right*. It folds over multiple lists as follows:

```
(fold-right (lambda (x y z) (list 'op x y z))  
            '0  
            '(a b c)  
            '(d e f))  
=> '(op a d (op b e (op c f 0)))
```

Fold-right is a bit trickier to write than *fold-left*. Do you want to give it a try before looking at the

zenstyle programming

following code?

```
(define (fold-right f b . a*)
  (letrec
    ((foldr
      (lambda (a* r)
        (cond ((null (car a*)) r)
              (t (foldr (cdr-of a*)
                        (apply f (append (car-of a*)
                                         (list r)))))))))
    (cond ((null a*) (bottom 'too-few-arguments))
          (t (foldr (map reverse a*) b))))))
```

Fold-right uses **append** in its general case. Is this a problem? If so, what can you do about it? If not, why not? (Q4)

4.7 substitute variables

The *substitute* function replaces variables in forms with values stored in a given environment:

```
(define env '((x . foo) (y . bar)))
(substitute '(cons x y) env) => '(cons foo bar)
```

Substitute is rather straight forward to implement. You may try it before reading ahead. Hint: its code is similar to *replace* [page 30].

```
(define (substitute x env)
  (letrec
    ((value-of
      (lambda (x)
        (let ((v (assq x env)))
          (cond (v (cdr v))
                (t x))))))
    (subst
      (lambda (x)
        (cond ((null x) ())
              ((atom x) (value-of x))
              (t (cons (subst (car x))
                       (subst (cdr x)))))))
    (subst x)))
```

What happens if you remove the clause **((null x) ())** from the **cond** of *subst*?

Why can *substitute* not be used to substitute variables with values in the bodies of lambda forms? Hint: this is related to beta reduction [page 59]. (Q5)

5. sorting

5.1 insertion sort

The *insert* function inserts an item into an ordered list. An ordered list is a list

$(x_1 \ x_2 \ \dots \ x_n)$

where a predicate p holds for each two consecutive members:

$(p \ x_i \ x_{i+1}) \Rightarrow :t$

for each i in $1..n-1$. Given a variadic predicate p ,

$(\text{apply } p \ x)$

applies to each list that is ordered under p .

Here is the code of *insert*:

```
(define (insert p x a)
  (letrec
    ((ins
      (lambda (a r)
        (cond ((or (null a) (p x (car a)))
              (append (reverse (cons x r)) a))
              (t (ins (cdr a) (cons (car a) r))))))
    (ins a ())))
```

A sorted list is constructed by successively inserting elements into an originally empty list:

```
(load ~nmath)
(insert < '#5 ()) => '(#5)
(insert < '#1 '#5) => '(#1 #5)
(insert < '#7 '#1 #5) => '(#1 #5 #7)
(insert < '#3 '#1 #5 #7) => '(#1 #3 #5 #7)
```

Note that only asymmetric predicates (like $<$) can impose an order on a list. For example, using the (symmetric) **neq** predicate to insert elements to a list does not order that list:

```
(load ~nmath)
(insert neq '#5 ()) => '(#5)
(insert neq '#1 '#5) => '(#1 #5)
(insert neq '#7 '#1 #5) => '(#7 #1 #5)
(insert neq '#3 '#7 #1 #5) => '(#3 #7 #1 #5)
```

A predicate is symmetric if $(p \ b \ a)$ follows from $(p \ a \ b)$ for each a and b . A predicate is asymmetric if this implication does not hold. Asymmetric predicates that are often used to order lists include $<$, $<=$, $>$, and $>=$.

zen style programming

The *isort* function inserts multiple members into an originally empty list, thereby effectively sorting those members:

```
(isort < '(#5 #1 #7 #3)) => '(#1 #3 #5 #7)
```

Here comes its code:

```
(require 'insert)
(define (isort p a)
  (letrec
    ((sort
      (lambda (a r)
        (cond ((null a) r)
              (t (sort (cdr a)
                       (insert p (car a) r))))))
    (sort a ())))
```

This algorithm is widely known as *insertion sort*.

How many times does *isort* have to apply its predicate *p* in order to sort a list of *n* elements, if:

- the elements are sorted in reverse order;
- the elements already are sorted;
- the elements are randomly distributed?

Do the above values differ by a wide margin? Does the result make insertion sort a practicable sorting algorithm or not? (**Q6**)

5.2 quicksort

Quicksort is probably one of the most efficient algorithms around. It uses a *divide and conquer* approach. This approach works by dividing the problem into smaller subproblems, solving them, and then re-assembling the result:¹⁰

```
(require 'partition)
(define (quicksort p a)
  (letrec
    ((sort
      (lambda (a)
        (cond ((or (null a) (null (cdr a))) a)
              (t (let ((p* (partition (lambda (x) (p (car a) x))
                                     (cdr a))))
                  (append (sort (cadr p*))
                          (list (car a))
                          (sort (car p*)))))
              (sort a))))
    (sort a))))
```

¹⁰ The approach works even in the real world. By separating the individuals of big masses of people, the people can easily be controlled by small, power-hungry groups. Advertisements and “news” are typical real-world divide and conquer tools. What can we do about them?

When quicksort sorts a list, it first divides that list into smaller lists, one containing members below a given threshold and one containing members above that threshold. The **trace** meta function can be used to look under the hood:

```
(trace sort) => :t
(quicksort < '#5 #1 #9 #3 #7))
+ (sort (#5 #1 #9 #3 #7))
+ (sort (#1 #3))           ; sort members <5
+ (sort ())               ; sort members <5 and <1
+ (sort (#3))             ; sort members <5 and >1
+ (sort (#9 #7))          ; sort members >5
+ (sort (#7))             ; sort members >5 and <9
+ (sort ())               ; sort members >5 and >9
=> '#1 #3 #5 #7 #9)
```

After sorting all partitions, *quicksort* re-assembles them using **append**:

```
(trace append) => :t
(quicksort < '#5 #1 #9 #3 #7))
+ (append () (#1) (#3))
+ (append (#7) (#9) ())
+ (append (#1 #3) (#5) (#7 #9))
=> '#1 #3 #5 #7 #9)
```

Note that *quicksort* itself never applies the predicate *p*. The list is sorted by partitioning it again and again:

```
(partition (lambda (x) (p '#5 x))
           '#1 #9 #3 #7))
=> '((#9 #7) (#1 #3))
```

After partitioning the list '#1 #9 #3 #7', the first partition contains only values below 5 and the second one only values above 5. These partitions are sorted recursively and then concatenated (together with the threshold value itself).

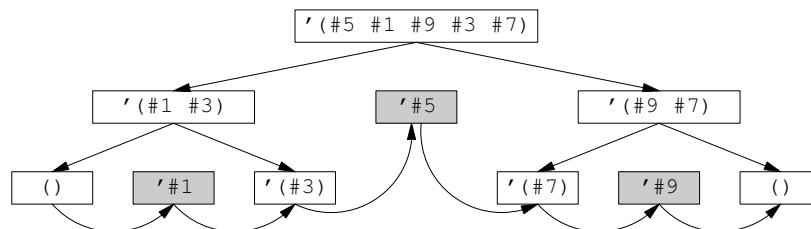


Fig. 1 – divide and conquer approach

Figure 1 illustrates the divide and conquer paradigm. The original list is recursively broken down into smaller lists until the smaller lists are trivial to sort, because they are either empty or contain only single values. The straight arrows indicate partitioning and the grey boxes contain threshold values. The curved arrows denote the **append** operations.

zen style programming

How does *quicksort* compare to *isort*? How does *quicksort* perform when sorting already sorted and reverse sorted input?

5.3 mergesort

The performance of both *quicksort* and *isort* depends on the data passed to them. This is, of course, not a desirable property for a sorting algorithm.

Mergesort is about as efficient as *quicksort* while its performance is constant. In environments without mutable data (like *zenlisp*), *mergesort* is even slightly more efficient than *quicksort*.

Here is its code:

```
(define (mergesort p a)
  (letrec
    ((split
      (lambda (a r1 r2)
        (cond ((or (null a)
                    (null (cdr a)))
              (list (reverse r2) r1))
              (t (split (cddr a)
                        (cdr r1)
                        (cons (car r1) r2)))))))
    (merge
      (lambda (a b r)
        (cond ((null a)
              (cond ((null b) r)
                    (t (merge a (cdr b) (cons (car b) r)))))
              ((null b)
              (merge (cdr a) b (cons (car a) r)))
              ((p (car a) (car b))
              (merge a (cdr b) (cons (car b) r)))
              (t (merge (cdr a) b (cons (car a) r))))))
    (sort
      (lambda (a)
        (cond ((or (null a)
                    (null (cdr a)))
              a)
              (t (let ((p* (split a a ())))
                    (merge (reverse (sort (car p*)))
                          (reverse (sort (cadr p*)))
                          ())))))
      (sort a)))
```

Mergesort's performance is constant because it *splits* its input rather than partitioning it. This is what the *split* function inside of *mergesort* does:

```
(split '#abcdef '#abcdef ()) => '(#abc #def)
```

As this example shows, the list is simply split in the middle, no matter which members it contains. Therefore *mergesort* always divides its input in an optimal way.

You may wonder why *split* actually splits the list in the middle. Would it not be easier to split the list by simply pushing its members to two other lists alternately? Like this:

```
(define (naive-split a r1 r2)
  (cond ((null a)
        (list r1 r2))
        ((null (cdr a))
         (list (cons (car a) r1) r2))
        (t (naive-split (cddr a)
                          (cons (car a) r1)
                          (cons (cadr a) r2)))))
(naive-split '#abcdef () ()) => '(#eca #fdb)
```

While this function is a bit simpler and possibly even a bit more efficient than *split*, it is also wrong. Can you explain why?

The answer becomes obvious when splitting less trivial data:

```
(define set '((#5 b) (#1 a) (#5 c) (#7 a) (#9 a) (#3 a)))
(split set set ())
=> '(((#5 b) (#1 a) (#5 c)) ((#7 a) (#9 a) (#3 a)))
(naive-split set () ())
=> '(((#9 a) (#5 c) (#5 b)) ((#3 a) (#7 a) (#1 a)))
```

Each member of the above set contain a numeric key and an additional symbol. Both splitting functions split the set evenly, but *split* preserves the order of the members while *naive-split* swaps some members.

While both splitting functions would work fine in *mergesort*, the naive variant would result in a sorting function that is *unstable*. An unstable sorting function sorts members just fine, but may swap members with equal keys in the process. Stability is a desirable property in sorting functions.

BTW, both *quicksort* and *mergesort* are only stable when non-strict predicates like `<=` and `>=` are used. They are unstable when strict predicates like `<` and `>` are passed to them. *Isort* is only stable when strict predicates are used.

Which of these two approaches do you consider advantageous? Why? (Q7)

Can you modify *isort* in such a way that it becomes stable when using non-strict predicates? (Q8)

You probably saw it coming: can you modify *quicksort* and *mergesort* in such a way that they become stable when using strict predicates? (Q8)

5.4 unsorting lists

When examining sorting functions, it would be nice to have some unsorted data. The *unsort*

zen style programming

function creates lists of unsorted natural numbers:

```
(unsort '(#1 #2 #3 #4 #5 #6 #7 #8 #9 #10) '#3)
=> '(#8 #5 #1 #7 #10 #9 #3 #2 #6 #4)
```

Its input must be a list of consecutive natural numbers, but they need not occur in any specific order:

```
(unsort '(#8 #5 #1 #7 #10 #9 #3 #2 #6 #4) '#5)
=> '(#5 #8 #1 #2 #3 #6 #7 #4 #10 #9)
```

The second argument of *unsort* is the “seed” — the first member of the list that will be unsorted. It must be a natural number that is less than the length of the given list.

Here is the code of *unsort*:

```
(require '~nmath)
(require 'nth)

(define (unsort a seed)
  (letrec
    ((remove-nth
      (lambda (a n r)
        (cond ((zero n)
              (cond ((null a) (reverse r))
                    (t (append (cdr a) (reverse r)))))
              (t (remove-nth (cdr a)
                             (- n '#1)
                             (cons (car a) r))))))
    (unsort4
      (lambda (a n k r)
        (cond ((zero k) (cons (car a) r))
              (t (unsort4 (remove-nth a n ())
                           (remainder (car a) k)
                           (- k '#1)
                           (cons (car (nth n a)) r))))))
    (unsort4 a seed (- (length a) '#1) ())))
```

When combined with *iota* [page 86], *unsort* can be used to create larger sets of unsorted numbers. Such sets are useful for examining the efficiency of sorting functions. The following definitions create some test sets:

```
(define sorted-set (iota '#1 '#100)) ; 1..100
(define reverse-set (reverse sorted-set)) ; 100..1
(define random-set (unsort sorted-set '#99)) ; random order
```

In combination with the **stats** meta function, sets like these can be used to test how sorting functions perform when sorting already sorted input, reverse sorted input and random input.

Figure 2 summarizes some results. The numbers in the table denote *reduction steps*. Each reduction, like the retrieval of the value of a variable or a function application, is one step.

For each algorithm, the steps needed to sort already sorted data, reverse sorted data, and random data are given. The fact that *quicksort* performs dramatically bad when sorting non-random data is in fact a flaw of the implementation, not a flaw of the algorithm.

size	isort			quicksort			mergesort		
	sorted	reverse	random	sorted	reverse	random	sorted	reverse	random
10	43,764	9,701	29,923	38,014	46,245	30,587	18,705	17,679	22,619
20	174,833	20,422	95,433	158,906	184,724	55,261	46,805	45,262	65,116
30	408,957	32,096	244,376	371,783	430,858	126,311	79,393	82,692	109,486
40	758,136	44,723	346,366	685,945	796,647	169,333	120,065	118,086	170,110
50	1,234,370	58,303	680,393	1,110,692	1,294,091	256,190	168,093	166,374	235,366
60	1,849,659	72,836	984,197	1,655,324	1,935,190	422,885	208,201	219,251	310,940
70	2,616,003	88,322	1,388,878	2,329,141	2,731,944	434,277	269,668	274,136	390,619
80	3,545,402	104,761	1,883,191	3,141,443	3,696,353	621,906	324,707	324,210	477,393
90	4,649,856	122,153	2,516,138	4,101,530	4,840,417	623,504	399,655	394,382	569,448
100	5,896,572	140,126	3,137,972	5,193,349	6,131,343	825,283	457,256	460,812	676,687

Fig. 2 – run times of sorting algorithms

The fact that the present implementation of *quicksort* performs as bad as *isort* when sorting already sorted data and equally bad when sorting reverse sorted data can be remedied by selecting a random threshold from the current partition (p^*).

Nevertheless the Quicksort algorithm *does* have a weakness that is caused by its dependence on the data to sort. Much research has been done in order to break this dependency, and in fact *efficient* implementations of Quicksort are the fastest sorting functions in practice.

Mergesort is much easier to implement efficiently than Quicksort because its performance does not depend on the data to sort¹¹ and it has an advantage in purely functional environments where data may not be mutated.

It is also worth noting that Insertion Sort, which performs abysmally on random and already-sorted sets, excels at sorting sets in reverse sorted order. In this case, it is even more efficient than the other two algorithms. In fact, the following algorithm is more suitable for inserting *small* amounts of data into a sorted list than the other two sorting algorithms:

```
(define (insert-small-set p small sorted)
  (isort p (append small (reverse sorted))))
```

For larger sets, *mergesort* should be used in the place of *isort* and **reverse** should be omitted. Feel free to explore reasonable limits for the size of the set to insert before switching to Mergesort becomes imperative.

When looking at the columns that compare the steps needed to sort random data (printed in

¹¹ However, *mergesort* exhibits a slight bias toward sorted data in the table. Can you explain this?

boldface characters in figure 2), you may notice that individual numbers say little about the qualities of the sorting algorithms. It is the *development* of these numbers as the sizes of the input sets increase which is interesting.

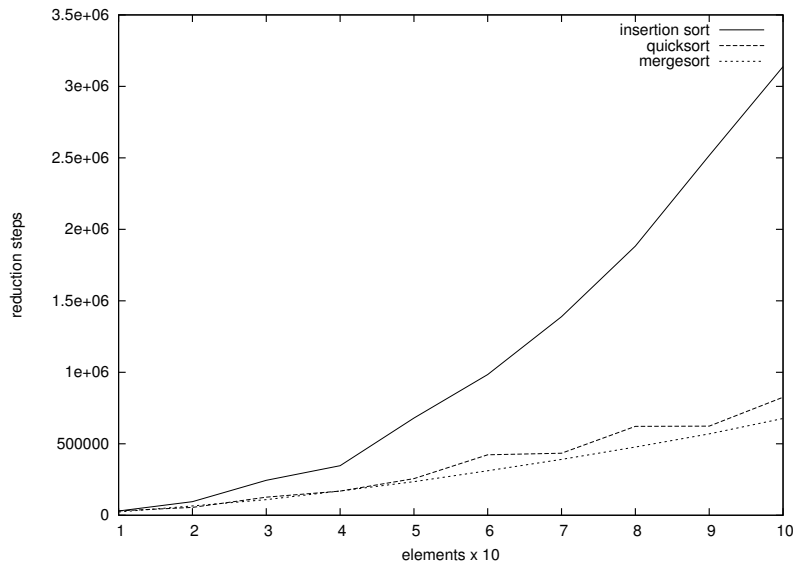


Fig. 3 – complexity of sorting algorithms

This development is called the *complexity* of a function. Figure 3 renders the complexity of the sorting functions discussed in this section as graphs. The x-axis reads the size of the random sets to sort, the y-axis the steps needed to sort a given set.

As can be seen in the graph, the run time of Insertion Sort grows quickly when the set size increases, while the run times of both Quicksort and Mergesort grow only slowly. Flat curves indicate high efficiency, steep curves indicate bad performance.

Complexity can be expressed without using curves by using the so-called *big-O notation*: $O(n \times 5)$ may be interpreted as “the runtime of the described algorithm multiplies by five when increasing its input by 1”. Big-O notation may describe both space and time requirements of an algorithm. In this section, we concentrate on time.

The average (time) complexity of Insertion Sort is $O(n^2/2)$, and the (average) complexity of Quicksort and Mergesort is $O(n \times \log(n))$. As we have seen, Quicksort has a worst-case performance of $O(n^2/2)$ (like Insertion sort), while the worst-case performance of Mergesort is (about) equal to its average performance.

The important part of the big-O notation is the formula itself, not their coefficients. For example, $O(n \times 10000)$ is generally *much* better than $O(1.1^n)$, even though the former has a much greater coefficient. For small values of n , the latter is indeed more efficient:

$$O(10 \times 10000) = 100000$$

$$O(1.1^{10}) = 2.5$$

At $n=150$, they almost break even:

$$O(150 \times 10000) = 1500000$$

$$O(1.1^{150}) = 1617717.8$$

But at $n=1000$, things look entirely different:

$$O(1000 \times 10000) = 100000000$$

$$O(1.1^{1000}) = 246993291800582633412408838508522147770973.3$$

$O(c \times n)$ (where c is constant) describes “linear” complexity, which is “very good”, while $O(c^n)$ indicates “exponential” complexity, which in most cases means that either the algorithm it describes is close to unusable or the problem it attempts to solve is *very* hard.

Figure 4 provides an overview over different classes of complexity, from best to worst.

formula	name
$O(c)$	constant
$O(c \times \log(n))$	logarithmic
$O(c \times n)$	linear
$O(c \times n^2)$	quadratic
$O(n^c)$	polynomial or geometric
$O(c^n)$	exponential

Fig. 4 – classes of complexity

BTW: which complexity does the *unsort* function have in the average case and in the worst case?
(Q9)

Do you think it can be improved?

6. logic and combinatoric functions

6.1 turning lists into sets

A *set* is a list of unique members. For instance, '(a b c) is a set, but '(a a b) is not because 'a occurs twice in the list. Just as a finger exercise: can you create a tail recursive function that converts a list to a set? E.g.:

```
(list->set '(a a b)) => '(a b)
```

Here is one possible solution:

```
(define (list->set a)
  (letrec
    ((l->s
      (lambda (a r)
        (cond ((null a)
              (reverse r))
              ((member (car a) r)
               (l->s (cdr a) r))
              (t (l->s (cdr a) (cons (car a) r))))))
    (l->s a ())))
```

6.2 union and intersection of sets

Using *list->set* computing the union of multiple sets is so simple that it is barely worth writing a function for it:

```
(require 'list-to-set)

(define (union . a)
  (list->set (apply append a)))
```

The intersection of sets is a bit trickier. It was discussed in depth earlier in this book [page 24]. Here comes a tail recursive variant using **fold**.

Of course you may try writing it yourself before reading ahead.

```
(define (intersection . a)
  (letrec
    ((intersection3 (lambda (a b r)
                      (cond ((null a)
                            (reverse r))
                            ((member (car a) b)
                             (intersection3 (cdr a) b (cons (car a) r)))
                            (t (intersection3 (cdr a) b r)))))
    (fold (lambda (a b)
           (intersection3 a b ()))
          (car a)
          a)))
```

6.3 find members with a given property

The *any* function checks whether a list contains at least one member with a given property:

```
(define (any p a)
  (cond ((null a) :f)
        (t (or (p (car a))
                 (any p (cdr a))))))
(any atom '(#ab #cd e)) => :t
```

The R⁶RS defines a generalized version of *any* called *exists*. It relates to *any* in the same way as *map-car* [page 44] relates to *map*: it accepts any positive number of list arguments and an *n*-ary function to be applied to them.

The following application of *exists* finds out whether there exists any position *i* in the lists *a*, *b*, *c* where *b_i* is the least value of the triple (*a_i*, *b_i*, *c_i*).

```
(exists (lambda (ai bi ci)
          (and (< bi ai)
               (< bi ci)))
  a
  b
  c)
```

Once again the *car-of* and *cdr-of* functions [page 67] prove helpful when turning a fixed-argument function into a variadic one:

```
(define (exists p . a*)
  (letrec
    ((exists*
      (lambda (a*)
        (cond ((null (car a*)) :f)
              (t (or (apply p (car-of a*))
                     (exists* (cdr-of a*)))))
      (exists* a*)))
```

BTW: *exists* is not a predicate. Can you explain way?

Exists can do something more interesting than checking whether a tuple with the given property exists: it can return that tuple:

```
(exists (lambda (ai bi ci)
          (and (< bi ai)
               (< bi ci)
               (list ai bi ci)))
  '(#3 #1 #4 #1 #5)
  '(#9 #2 #6 #5 #3)
  '(#5 #8 #9 #7 #9))
=> '(#5 #3 #9)
```

Because *exists* may return a value other than *:t* or *:f* when a non-predicate is passed to it, it is itself not a predicate.

6.4 verify properties

You may have observed that the *exists* function implements the *existential quantor*:

*There exists an **x**, **y**, ... for which (**p x y** ...).*

The R⁶RS also describes a function implementing the *universal quantor* (for all **x**, **y**, ...). Unsurprisingly it is called *for-all* and returns truth if the given predicate applies to *all* tuples that are formed by combining list elements at equal positions:

```
(for-all eq '#abcdef '#abcdef) => :t
```

When a non-predicate is passed to *for-all*, it returns the *last* tuple that could be formed (or **:f**):

```
(for-all (lambda (x y)
  (and (eq x y)
    (list x y)))
 '#abcdef
 '#abcdef) => #ff
```

Here is the code of *for-all*:

```
(define (for-all p . a*)
  (letrec
    ((forall*
      (lambda (a*)
        (cond ((null (car a*)) :t)
              ((null (cdr a*))
               (apply p (car-of a*)))
              (t (and (apply p (car-of a*))
                      (forall* (cdr-of a*)))))
        (forall* a*))))))
```

In which way does the behavior of the function change when you omit the clause

```
((null (cdr a*)) (apply p (car-of a*)))
```

from the above code? (**Q10**)

For-all returns **:t** when empty lists are passed to it and *exists* returns **:f** in this case. Do you think this makes sense? Why?

6.5 combinations of sets

The functions discussed in this section are a bit more complex, so if you need a break, this would be an excellent time to take one.

A *combination* of the **n**'th order (an **n**-combination) of a source set **a** is an **n**-element set of elements from **a**. For example, '#ab is a 2-combination of the set '#abc.

The *combine* function generates *all* possible **n**-element combinations without repetition from a given set:

```
(combine '#2 '#abc) => ' (#ab #ac #bc)
```

*Combine** creates combinations *with* repetition:

```
(combine* '#2 '#abc) => ' (#aa #ab #ac #bb #bc #cc)
```

Because the algorithms for these functions are very similar, they are both implemented in the higher-order *combine3* function. The differing part of the algorithms is passed to *combine3* as an argument.

Combinations of **n** elements with repetition are created as follows:

If **n=0**, the set of combinations is empty:

```
(combine '#0 x) => () ; for any x
```

If **n=1**,

```
(combine '#1 x) -> (map list x) ; for any x
```

For **n>1**, combinations are created recursively. The first step in this process is to create all possible non-empty tails of the source set. This is what the *tails-of* function does:

```
(define (tails-of set)
  (cond ((null set) ())
        (t (cons set (tails-of (cdr set))))))

(tails-of '#abcd) => ' (#abcd #bcd #cd #d)
```

N-combinations with repetition are created from the result of *tails-of* as follows: the head of each sublist is attached to all **n-1**-element combinations of the same sublist. The following example outlines this process for **n=2** and **set=' #abcd**:

head	tail	1-combinations	result
#a	#abcd	(#a #b #c #d)	(#aa #ab #ac #ad)
#b	#bcd	(#b #c #d)	(#bb #bc #bd)
#c	#cd	(#c #d)	(#cc #cd)
#d	#d	(#d)	(#dd)

The concatenation of the **result** column is the set of all 2-combinations of '**#abcd**:

```
' (#aa #ab #ac #ad #bb #bc #bd #cc #cd #dd)
```

In the above example, *combine* calls itself with **n=1**, so this application is handled by a trivial case. Here is what happens when combinations of a higher order are generated (the example creates the 3-combinations of '**#abcd**):

zen style programming

head	tail	2-combinations	result
#a	#abcd	(#aa #ab #ac #ad #bb #bc #bd #cc #cd #dd)	(#abc #abd #acc #acd #add #aaa #aab #aac #aad #abb)
#b	#bcd	(#bb #bc #bd #cc #cd #dd)	(#bbb #bbc #bbd #bcc #bcd #bdd)
#c	#cd	(#cc #cd #dd)	(#ccc #ccd #cdd)
#d	#d	(#dd)	(#ddd)

The 2-combinations that are appended to the head of each tail set are created in the same way as in the previous example. Because **n** decreases, the trivial case is finally reached.

Here follows the code of the *combine2* function which creates combinations with repetition:

```
(require '~nmath)

(define (combine2 n set)
  (cond
    ((zero n) ())
    ((one n) (map list set))
    (t (apply
        append
        (map (lambda (tail)
              (map (lambda (sub)
                    (cons (car tail) sub))
                    (combine2 (- n '#1) tail))))
            (tails-of set))))))
```

The general case of the function contains two nested **maps** which create combinations recursively. The list passed to the outer **map** is the tail set of the current source set. The inner **map** recurses to create combinations of a lower order and then attaches these combinations to the head of *tail*. Finally, **applying append** flattens the result of the outer map.

Combinations without repetition are created in basically the same way as combinations with repetition. The only difference is that items from the source set may not be reused, so they have to be removed from the set before creating lower-order combinations. Here is how it works:

head	tail	1-combinations	result
#a	#bcd	(#b #c #d)	(#ab #ac #ad)
#b	#cd	(#c #d)	(#bc #bd)
#c	#d	(#d)	(#cd)

So instead of the entire set only the **cdr** part should be passed to *combine2* when recursing. The modification is so trivial that the function doing the **cdr** operation is passed to *combine3* as an additional argument. Differences to *combine2* are rendered in boldface characters:

```
(define (combine3 n set rest)
  (lambda (n set)
    (cond
      ((zero n) ())
      ((one n) (map list set))
      (t (apply
          append
          (map (lambda (tail)
                 (map (lambda (sub)
                        (cons (car tail) sub))
                    (combine3 (- n '#1) (rest tail) rest)))
                (tails-of set)))))))
```

When *rest*=**cdr** is passed to *combine3*, it computes combinations without repetition. To compute combinations with repetition, the identity function **id**¹² is passed to it instead. So *combine* and *combine** can be defined as follows:

```
(define (combine n set)
  (combine3 n set cdr))

(define (combine* n set)
  (combine3 n set id))
```

Which kind of recursion does *combine3* use?

Do the complexities of *combine* and *combine** differ? Estimate their complexities. (Q11)

6.6 permutations of sets

A *permutation* differs from a *combination* in the respect that order does not matter in combinations but does matter in permutations. Hence the sequences '#ab and '#ba denote the same combination but different permutations. Both '#ab and '#ba *combine* a with b, but '#ab is a permutation of '#ba (in this case the only one).

Like combinations, permutations can be created with and without repetition. The *permute* and *permute** functions introduced in this section create all possible permutations of a set with and without repetition:

```
(permute '#2 '#abc) => ' (#ab #ba #ac #ca #bc #cb)
(permute* '#2 '#abc) => ' (#aa #ab #ac #ba #bb #bc #ca #cb #cc)
```

We will start with a function that creates *n*-permutations without repetition from source sets of the size *n*, e.g.:

```
(permutations '#abc) => ' (#abc #acb #bca #bac #cab #cba)
```

The trivial cases of this function are simple. 0-permutations are empty:

¹² **Id** is defined as `(lambda (#x x)).`

zen style programming

```
(permutations ()) => ()
```

and 1-permutation are equal to a set containin the source set:

```
(permutations '(x)) → (list '(x)) ; for any atom x
```

Permutations are created in almost the same was as combinations, but because order does matter in permutations, each of the elements of the source set has to occupy each position once. This is easily achieved by rotating the members of the set:

```
'#abcd  
'#bcda  
'#cdab  
'#dabc
```

The *rotate* function rotates a set by one position:

```
(define (rotate x)  
  (append (cdr x) (list (car x))))
```

Using *rotate*, creating *all* rotations is easy:

```
(define (rotations x)  
  (letrec  
    ((rot (lambda (x n)  
            (cond ((null n) ())  
                  (t (cons x (rot (rotate x)  
                                   (cdr n)))))))  
    (rot x x)))
```

```
(rotations '#abcd) => '(#abcd #bcda #cdab #dabc)
```

The *permutations* function itself looks pretty much like *combine3* [page 83]. It just has different trivial cases and uses *rotations* in the place of *tails-of*:

```
(define (permutations set)  
  (cond  
    ((null set) ())  
    ((null (cdr set)) (list set))  
    (t (apply append  
              (map (lambda (rotn)  
                    (map (lambda (x)  
                        (cons (car rotn) x)  
                          (permutations (cdr rotn))))  
                    (rotations set))))))
```

While this function works fine, it can only create permutations whose order equal to the size of the source set. Permutations of a lower order can be computed by creating the combinations of the same order first and then permutating and appending the results:

```
(combine '#2 '#abc) => '(#ab #ac #bc)  
(map permutations '(#ab #ac #bc)) => '((#ab #ba) (#ac #ca) (#bc #cb))
```

```
(apply append '((#ab #ba) (#ac #ca) (#bc #cb)))  
=> '(#ab #ba #ac #ca #bc #cb)
```

This is exactly what the following implementation of *permute* does:

```
(require 'combine)  
  
(define (permute n set)  
  (apply append (map permutations (combine n set)))))
```

Permutations with repetition are created in a similar way as combinations with repetition (see *combine2*, page 82). The only difference is that the whole source set is passed along in each **map**:

```
(define (permute* n set)  
  (cond  
    ((zero n) ())  
    ((one n) (map list set))  
    (t (apply append  
              (map (lambda (x)  
                    (map (lambda (sub)  
                          (cons x sub)  
                                (permute* (- n '#1) set)))  
                      set))))))
```

What complexity does the *permutations* function have?

The 2-permutations of a 2-element set are equal to the rotations of that set:

```
(permutations '#xy) => '(#xy #yx)  
(rotations '#xy) => '(#xy #yx)
```

So we could add a third trivial case to the *permutations* function:

```
((null (cddr set)) (rotations set))
```

Where would you insert this case? Would this modification make sense? Would it improve the run time of *permutations*? Would it change its complexity? (**Q12**)

7. math functions

7.1 sequences of numbers

The *iota* function creates sequences of integer numbers:

```
(iota '#1 '#10) => '(#1 #2 #3 #4 #5 #6 #7 #8 #9 #10)
(iota '#-3 '#-1) => '(-3 -2 -1)
```

In combination with **map** it can be used to create various kinds of sequences:

```
(map (lambda (x) (* x x))
     (iota '#1 '#5))
=> '(#1 #4 #9 #16 #25)
(require '~combine)
(map (lambda (x) (combine x '#abcde))
     (iota '#0 '#5))
=> '(()
    (#a #b #c #d #e)
    (#ab #ac #ad #ae #bc #bd #be #cd #ce #de)
    (#abc #abd #abe #acd #ace #ade #bcd #bce #bde #cde)
    (#abcd #abce #abde #acde #bcde)
    (#abcde))
(map (lambda (x) (length (combine* x '#abcd)))
     (iota '#1 '#10))
=> '(#4 #10 #20 #35 #56 #84 #120 #165 #220 #286)
```

The implementation of *iota* is much simpler than many of its applications:

```
(require '~imath)

(define (iota lo hi)
  (letrec
    ((j (lambda (x r)
          (cond ((< x lo) r)
                (t (j (- x '#1) (cons x r)))))))
    (j (integer hi) ())))
```

Can you use *iota* to estimate the complexities of some functions like *combine**, *permute**, *unsort*, or *iota* itself?

7.2 fast factorial function

The *factorial* function computes **n!** (“**n** factorial”), or

```
(* 1 2 ... n)
```

However, it uses an algorithm that is more efficient than just multiplying the numbers in sequence. It can compute values like 100! in reasonable time, even on a purely symbolic system like zenlisp.

The algorithm is called *recursive product*.

The code follows immediately. Can you see why it is more efficient than the naive approach?

```
(require '~nmath)

(define (factorial n)
  (letrec
    ((r* (lambda (n m)
          (cond ((< m '#2) n)
                (t (let ((q (quotient m '#2)))
                    (* (r* n q)
                       (r* (+ n q) (- m q)))))))
    (r* '#1 (natural n))))
```

The *factorial* function is *fat recursive*: it has the same complexity as a structural recursive function, although it does not process a recursive structure. This is generally considered a bad idea, but in this particular case, it actually improves the run time of the function.

Factorial uses a divide and conquer approach [see page 70]. Given the value **n**, it computes the products of $1..n/2$ and $n/2+1..n$ first and then multiplies them. It uses the same number of multiplications as the “naive” function (one less in fact), but it avoids to multiply large numbers as long as possible, as can be seen in the following table:

naive approach	recursive product
(f '#10)	(factorial '#10)
+ (* #1 #1)	+ (* #1 #2)
+ (* #2 #1)	+ (* #4 #5)
+ (* #3 #2)	+ (* #3 #20)
+ (* #4 #6)	+ (* #2 #60)
+ (* #5 #24)	+ (* #6 #7)
+ (* #6 #120)	+ (* #9 #10)
+ (* #7 #720)	+ (* #8 #90)
+ (* #8 #5040)	+ (* #42 #720)
+ (* #9 #40320)	+ (* #120 #30240)
+ (* #10 #362880)	

The output is created by tracing ***** while computing 10!.¹³

There is an even simpler (and interestingly more efficient) method for computing **n!** in zenlisp. This method does not use recursion at all. Hint: it does use a function from this section. (Q13)

¹³ The naive function is: (define (f x) (cond ((zero x) '#1) (t (* x (f (- x '#1)))))).

7.3 integer factorization

The *factors* function computes the constituent prime factors of a given integer:

```
(factors '#123456789) => '((#3803 #1) (#3607 #1) (#3 #2))
```

It returns a list of two-element lists where each sublist should be read as car^{cadr} , so the above integer factors into

$$3383^1 \times 3607^1 \times 3^2 = 3383 \times 3607 \times 3 \times 3$$

Factors may take a while to complete when computing the factors of primes, coprimes, or integers composed of rather large factors, like the above. It actually attempts to divide a given number *n* by {2,3,5,...,*sqrt*(*n*)} and memorizes the quotients found:

```
(require '~nmath)

(define (factors n)
  (letrec
    ((quotient+exponent
      (lambda (n m)
        (letrec
          ((div (lambda (n m r)
                  (let ((qr (divide n m)))
                    (cond ((zero (cadr qr))
                          (div (car qr) m (+ '#1 r)))
                          (t (cons n r)))))))
         (div n m '#0))))
    (add-expt
      (lambda (b e r)
        (cond ((zero e) r)
              (t (cons (list b e) r)))))
    (factorize
      (lambda (n d r)
        (let ((lim (sqrt n)))
          (letrec
            ((factorize3
              (lambda (n d r)
                (let ((rest/exp (quotient+exponent n d)))
                  (let ((q (car rest/exp))
                        (e (cdr rest/exp)))
                    (cond
                     ((< q '#2) (add-expt d e r))
                     ((> d lim) (add-expt n '#1 r))
                     (t (factorize3
                        q
                        (cond ((= d '#2) '#3)
                            (t (+ d '#2)))
                        (add-expt d e r)))))))
                 (factorize3 n d r))))))
        (cond
         ((< n '#1) (bottom 'operand-not-positive n))
         ((= n '#1) '#1)
         (t (factorize n '#2 ()))))))
```

The function is pretty straight-forward. Its helper function *quotient+exponent* returns the quotient that remains when dividing **n** by **m** a given number of times. The number of divisions is returned as the “exponent” part. For example:

```
(quotient+exponent '#24 '#2) => '(#3 . #3)
```

24 can be divided by 2 three times, so 3 is returned as exponent in the cdr part of the result. $24/2^3 = 3$, so 3 is also returned as quotient in the car part.

```
(quotient+exponent '#24 '#3) => '(#8 . #1)
```

24 can be divided by 3 one time, leaving a quotient of 8.

```
(quotient+exponent '#24 '#9) => '(#24 . #0)
```

24 cannot be divided by 9 at all (zero times, leaving a “quotient” of 24).

Add-expt adds a factor to the result, but only if the exponent is non-zero.

Factorize computes the limit of *sqrt(n)* and then iterates through the list of possible divisors.

Can you imagine why the square root of the integer to factor is chosen for the upper limit?

There does not seem to be a really efficient algorithm for factorizing large integers. If there was one, computer-based cryptology might very well become a lost art. This is not meant to stop you from searching for a better algorithm, though.

7.4 partitioning integers

A (number-theoretic) *partition* of an integer **n** is a sum of integers that adds up to **n**. For instance, these are the partitions of 4:

4 3 + 1 2 + 2 2 + 1 + 1 1 + 1 + 1 + 1

Creating partitions is similar to creating permutations. The only difference is that elements of a partition may be split in two, e.g. 3 may be split into 2 and 1, and 2 may be split into 1 and 1.

The following code is based on *permutations* [page 84]. Instead of attaching the head of a set to each permutation of its rest, it attaches each value **i** of the interval *1..n* to the partitions of **n-i**, e.g.:

for n=4

- 1 is attached to the partitions of 3;
- 2 is attached to the partitions of 2;
- 3 is attached to the partitions of 1;
- 4 is attached to the partitions of 0.

The trivial cases handle the values of zero (giving the empty partition) and one (giving a partition of 1). Partitions are represented as lists, so the partitions of 4 would be written as

zen style programming

```
'((#4) (#3 #1) (#2 #2) (#2 #1 #1) (#1 #1 #1 #1)).
```

Here is the code (but feel free to try to develop it on your own before reading ahead):

```
(require '~nmath)
(require 'iota)

(define (part n)
  (cond
    ((zero n) '())
    ((one n) '(#1))
    (t (apply append
      (map (lambda (x)
        (map (lambda (p) (cons x p))
              (part (- n x))))
            (iota '#1 n))))))
```

Let us see how it performs:

```
(part '#4)
=> '((#1 #1 #1 #1) (#1 #1 #2) (#1 #2 #1) (#1 #3) (#2 #1 #1) (#2 #2) (#3 #1) (#4))
```

Looks fine except that the list is in reverse order (which is easy to fix) and that it contains some duplicates (in boldface characters). So maybe we need a solution that is based on combinations rather than permutations? Before we dive to deep into this idea: is there something obvious about the above result?

Well?

Exactly: all the non-duplicate partitions are in descending order (but not in *strict* descending order), so we can filter them. This is what the *make-partitions* function does:

```
(define (make-partitions n)
  (letrec
    ((filter-descending
      (lambda (p)
        (cond ((null (cdr p)) p)
              ((apply >= (car p))
               (cons (car p) (filter-descending (cdr p))))
              (t (filter-descending (cdr p))))))
    (reverse (filter-descending (part n)))))
```

And indeed:

```
(make-partitions '#4) => '((#4) (#3 #1) (#2 #2) (#2 #1 #1) (#1 #1 #1 #1))
```

Why are we not using *filter* [page 42] to filter the results? (Q14)

Give a reasonable upper limit for the number of partitions of 100. Why is *make-partitions* not such a great help in this process?

7.5 exploring the limits of computability

Have a look at the following function:

```
(define (s x) (+ '#1 x))
```

The *s* function is much like **succ** [page 276], but increments whole natural numbers instead of just single digits:

```
(s '#0) => '#1
(s (s '#0)) => '#2
(s (s (s '#0))) => '#3
```

Using this function, the addition of two numbers **a** and **b** could be defined as the **b**-fold application of *s* to **a**:

$$a+b := \underbrace{(s \dots (s a)) \dots}_{b \text{ times}}$$

In the same way the multiplication of **a** and **b** can be expressed as the **b**-fold application of the “plus” operator, and **a** raised to the power of **b** can be expressed as the **b**-fold application of multiplication:

$$a*b := \underbrace{a + \dots + a}_{b \text{ times}} \qquad a^b := \underbrace{a * \dots * a}_{b \text{ times}}$$

The game does not end here. The **b**-fold application of the exponentiation operator to **a** is called a *power tower* of the height **b**:

$$a^{^b} := \underbrace{a \begin{matrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{matrix} a}_{b \text{ times}}$$

Typical notations for the power tower are $a^{^b}$ and $a^{(4)}b$. The former operator is pronounced “power power” and the latter is pronounced “hyper-4”. The *hyper operator* is in fact a generalization of all of the operations described above:

hyper notation	equivalent form
$a^{(0)}b$	$(s a)$ for any value of b
$a^{(1)}b$	$a + b$
$a^{(2)}b$	$a * b$
$a^{(3)}b$	a^b
$a^{(4)}b$	$a^{^b}$

zen style programming

Once again, the implementation of the function looks more innocent than its implications will turn out to be:

```
(require '~nmath)

(define (hyper n a b)
  (cond ((equal n '#0) (+ '#1 a))
        ((equal n '#1) (+ a b))
        ((one b)      a)
        ((equal n '#2) (* a b))
        ((equal n '#3) (expt a b))
        ((equal n '#4) (expt a (hyper n a (- b '#1))))
        ((> n '#4)     (hyper (- n '#1) a (hyper n a (- b '#1))))))
```

Now that we have learned that there is a hyper-4 operator, why should there not be a hyper-5 operator, a tower of power towers? Or a hyper-6 operator or, for that matter, a hyper-100 operator?

We will see.

One thing is certain: no matter which argument of *hyper* is increased, its values grow really fast if *n* is at least 4:

```
(hyper '#4 '#1 '#3) = 1^1 = 1^1 = 1 (1 digit)
(hyper '#4 '#2 '#3) = 2^2 = 2^2 = 2 (1 digit)
(hyper '#4 '#3 '#3) = 3^3 = 3^3^3 = 3^27 = 7625597484987 (13 digits)
(hyper '#4 '#4 '#3) = 4^3 = 4^4^4 = 4^256 = ... (155 digits)
(hyper '#4 '#5 '#3) = 5^3 = 5^5^5 = 5^3125 = ... (2185 digits)
(hyper '#4 '#6 '#3) = 6^3 = 6^6^6 = 6^46656 = ... (36306 digits)
```

Iterating the first factor of *hyper* leads to impressive growth. Even $4^{(4)}3$ (or **(hyper '#4 '#4 '#3)**) has a result that is *far* larger than the number of atoms in the known universe (which is about 10^{80}).

But the above is only the beginning. As we have seen in the section about the complexity of functions [page 75], iterating the exponent yields *much* faster growth than iterating the base of a power. So:

```
(hyper '#4 '#3 '#1) = 3
(hyper '#4 '#3 '#2) = 3^3 = 27
(hyper '#4 '#3 '#3) = 3^3 = 3^3^3 = 3^27 = 7625597484987
(hyper '#4 '#3 '#4) = 3^4 = 3^3^3^3 = 3^3^27 = 3^7625597484987
(hyper '#4 '#3 '#5) = 3^5 = 3^3^3^3^3 = 3^3^3^27 = 3^3^7625597484987
```

$3^{(4)}4 = 3^{7625597484987}$ is a number with about 3,638,334,640,024 digits. That is 3.6 *trillion* digits. If you print this number using a really small font and squeeze 100,000 digits on a page, you will still need 36 million pages. And $3^{(4)}5$ is 3 raised to the power of *that* number.

But even this growth is shallow compared to iterating the order of the hyper operator itself:

```
(hyper '#1 '#3 '#3) = 3+3 = 6
(hyper '#2 '#3 '#3) = 3*3 = 9
(hyper '#3 '#3 '#3) = 3^3 = 27
(hyper '#4 '#3 '#3) = 3^^3 = 3^3^3 = 3^27 = 7625597484987
(hyper '#5 '#3 '#3) = 3^^^3 = 3^^3^^3 = 3^^7625597484987
```

$3^{(5)}3$ is a power tower of the height 7,625,597,484,987. This is a tower of *7.6 trillion stories*:

$$\left. \begin{array}{c} 3 \\ \vdots \\ 3 \end{array} \right\} 7,625,597,484,987 \text{ times}$$

Such numbers are *way* beyond human comprehension, and the difference between one result and the next keeps *increasing* exponentially. This is why this kind of growth is called *hyper-exponential* growth.

Do you think that functions with hyper-exponential complexity may have any uses in practice?

Reduce $3^{(6)}3$ to lower-order operations as far as you can get. Attempt to describe the size of $3^{(6)}3$. (Q15)

Figure out all sets of arguments of *hyper* for which your computer will finally return a value. Just trying it is obviously not an option, so you will have to do some mental work, but this is what computing science is about.

Do you think that buying a faster computer with more memory would extend the above set?

What does $2^{(100)}2$ evaluate to? Yes, this is a trick question.

7.6 transposing matrixes

This chapter closes with a useful little one-liner that transposes a matrix. The matrix is stored as a list of rows. The function swaps columns and rows:

```
(transpose '(#abc #def)) => '(#ad #be #cf)
(transpose '(#ad #be #cf)) => '(#abc #def)
```

Here is the code:

```
(define (transpose x) (apply map list x))
```

Can you explain how it works?

8. data structures

8.1 generators

A generator is a data structure that generates a series of values. However, the concept of a generator seems to imply some mutable state, as the following example illustrates by means of a generator that creates the natural numbers:

```
(g) => '#1  
(g) => '#2  
(g) => '#3  
...
```

Each time **g** is called it returns a different value, so **g** cannot be a function in the strict sense. Yet it is possible to implement generators in **zenlisp** in a purely functional way. This is how it works:

```
(define (generator start step)  
  (lambda ()  
    (cons start  
          (generator (step start) step))))  
  
(define (value g) (car g))  
(define (next g) ((cdr g)))
```

The *generator* function is a higher order function that returns a generator. When called, the generator delivers a data structure consisting of the initial value *start* and another generator carrying **(step start)** as its value.

The structure is indefinitely recursive, but nevertheless finite. This is because no generator is reduced until requested. For this reason a generator is also called a *lazy* structure. Here is *generator* in action:

```
(load ~nmath)  
(generator '#1 (lambda (x) (+ '#1 x)))  
=> {closure ()}  
(**) => '(#1 . (closure ()))  
(next **) => '(#2 . (closure ()))  
(next **) => '(#3 . (closure ()))  
...
```

The ****** operator always contains the most recent toplevel result, so **(next **)** picks up the previous result and runs its embedded generator. Of course, ****** itself is stateful and works only in interactive computations, so here is the same example without it:

```
(let ((g ((generator '#1 (lambda (x) (+ '#1 x))))))  
  (let ((x (value g))  
        (g (next g)))  
    (let ((y (value g))  
          (g (next g)))
```

```
(let ((z (value g))
      (g (next g)))
      (list x y z))))
=> '(#1 #2 #3)
```

Each application of the form **(next g)** appears to have a side effect, because it delivers a new value, but what it really does is to map the current generator to a new one, so it is in fact an ordinary function.

Here is what happens “under the hood”:

```
(define (inc x) (+ '#1 x))
(generator '#1 inc)
=> (lambda () (cons '#1 (generator (inc '#1 inc))))
(**) => '(#1 . (lambda () (cons '#2 (generator (inc '#2 inc)))))
(next **) => '(#2 . (lambda () (cons '#3 (generator (inc '#3 inc)))))
(next **) => '(#3 . (lambda () (cons '#4 (generator (inc '#4 inc)))))
```

Can you create a generator that produces the tails of a list? What happens when the end of the list is reached? Can you improve the *generator* function in such a way that it handles such situations more gracefully? (Q16)

How are generators related to lists? What are their differences and what do they have in common?

8.2 streams

Basically streams are refined generators. Here is the implementation of the *stream* function which implements the *stream* data type:

```
(define (stream v first filter rest lim final)
  (letrec
    ((find
      (lambda (x)
        (cond ((lim x) x)
              ((filter (first x)) x)
              (t (find (rest x))))))
     (make-stream
      (lambda (v)
        (lambda ()
          (let ((nf (find v)))
            (cond ((lim nf) final)
                  (t (cons (first nf)
                           (make-stream (rest nf))))))))))
    ((make-stream v))))
```

The *v* variable of *stream* has the the same function as the *start* variable of *generator* [page 94] and *rest* is equivalent to *step*. The *make-stream* subfunction is basically equal to *generator*, with some features added.

Using *stream* a generator that produces natural numbers can be created this way:

zen style programming

```
(stream '#1
  id
  (lambda (x) :t)
  (lambda (x) (+ '#1 x))
  (lambda (x) :f)
:f)
```

The *first* variable is bound to a function that preprocesses each value of the stream before returning it. Because there is nothing to do in this example, the identity function is passed to *stream*.

The *filter* variable binds to a predicate that must reduce to truth for each member of the stream that is to be generated. The above filter just passes through all members.

Lim binds to a predicate checking for the end of the stream. The above predicate returns constant falsity, so the stream is (potentially) infinite. The value of *final* is to be returned when (**lim x**) returns truth for some **x**.

Predicates returning constant truth and falsity are common in streams, so they are defined as follows:

```
(define (all x) :t)
(define (none x) :f)
```

The *next* and *value* functions are the same as in the *generator* code:

```
(define (value s) (car s))
(define (next s) ((cdr s)))
```

Using these abbreviations, the above stream of natural numbers can be created in a more comprehensible way:

```
(stream '#1 id all (lambda (x) (+ '#1 x)) none :f)
```

This definition returns a stream “starting at *one*, returning the *identity* of *all* members, *incrementing members by one*, and having *no limit*”. The “final” value does not matter in this case because the limit is *none*.

Note that (unlike *generator*) stream functions return streams immediately and not functions returning streams:

```
(stream '#1 id all (lambda (x) (+ '#1 x)) none :f)
=> '(#1 . {closure ()})
(next **) => '(#2 . {closure ()})
(next **) => '(#3 . {closure ()})
...
```

A stream delivering the members of a list would be created as follows:

```
(stream '#abc car all cdr null :f)
=> '(a . {closure ()})
(next **) => '(b . {closure ()})
(next **) => '(c . {closure ()})
(next **) => :f
```

In fact, this expression is so useful that we will give it a name:

```
(define (list->stream v)
  (stream v car all cdr null :f))
```

Stream->list is the reverse operation of *list->stream*. It collects the members of a stream and places them in a list.

```
(define (stream->list s)
  (letrec
    ((s->l
      (lambda (s lst)
        (cond (s (s->l (next s)
                       (cons (value s) lst)))
              (t (reverse lst))))))
    (s->l s ())))
```

BTW, why is it not a good idea to convert a stream of natural numbers — like the one defined above — to a list?

Here follow some higher order functions that can be applied to streams. Most of these functions have exact counterparts in the list domain.

The *stream-member* function locates the first member satisfying the predicate *p* in the given stream. When no such member is found, the default value *d* is returned instead:

```
(define s (list->stream '(#a b #c d)))
(stream-member atom s :f) => '(b . {closure ()})
(stream-member null s :f) => :f
```

Here is the code of *stream-member*. Note that it uses *d* for both detecting the end of the stream and indicating the end of the stream:

```
(define (stream-member p s d)
  (cond ((eq s d) d)
        ((p (value s)) s)
        (t (stream-member p (next s) d))))
```

The *pass* function is another convenience function. It is used to indicate that an embedded stream should be considered to be exhausted when it returns **:f**. However, this is merely a convention used here. Any other value could be used to indicate the end of a stream.

```
(define pass not)
```

Map-stream is like **map**, but works on streams:

```
(require ~nmath)
(map-stream (lambda (x) (* x x))
  (stream '#1 id all (lambda (x) (+ '#1 x)) none :f))
=> '(#1 . {closure ()})
(next **) => '(#4 . {closure ()})
(next **) => '(#9 . {closure ()})
(next **) => '(#16 . {closure ()})
...
```


The *map-stream* function can be implemented using a single invocation of *stream*:

```
(define (map-stream f s)
  (stream s (lambda (s) (f (value s))) all next pass :f))
```

The stream created by it “starts at the value of *s*, returns *f* applied to *all* members, fetches *next* members from the original stream, *passes* end-of-stream detection to the original stream and returns *:f* as its final value.

So *map-stream* basically constructs a “stream around a stream”. The outer stream fetches values from the inner stream and applies a function to each value before returning it. In the above example, the inner stream still creates natural numbers while the outer stream — which is created by *map-stream* — generates squares.

The *filter-stream* function applies a filter to a stream, so that only members with specific properties are generated. It may be considered some kind of “internal *stream-member*” function. Indeed it implemented internally by the *find* function of *stream*. Again, the code is trivial:

```
(define (filter-stream p s)
  (stream s value p next pass :f))
```

Here is how it works:

```
(filter-stream atom (list->stream '(a #b c #d e)))
=> '(a . {closure ()})
(next **) => '(c . {closure ()})
(next **) => '(e . {closure ()})
(next **) => :f
```

Of course, stream functions can be combined:

```
(require ~nmath)
(map-stream (lambda (x) (* x x))
  (stream '#1 id all (lambda (x) (+ '#1 x)) none :f))
=> '(#1 . {closure ()})
(next **) => '(#4 . {closure ()})
(filter-stream even **)
=> '(#4 . {closure ()})
=> '(#16 . {closure ()})
(next **) => '(#36 . {closure ()})
(filter-stream (lambda (x) (zero (remainder x '#7))) **)
=> '(#196 . {closure ()})
(next **) => '(#784 . {closure ()})
(next **) => '(#1764 . {closure ()})
...
```

In this example *map-stream* once again creates a stream of square numbers. Then *filter-stream* filters all even numbers from that stream. Finally, another filter extracts all numbers that are divisible by 7 from the resulting stream. So the final stream generates even square numbers that are divisible by 7.

The last function introduced here appends two streams by wrapping the first stream around the second. When the first stream is exhausted, it returns the second one as its final value:

```
(define (append-streams s1 s2)
  (stream s1 value all next pass s2))
```

Here is *append-streams* in action:

```
(stream->list
  (append-streams (list->stream '#hello-)
                  (list->stream '#world!)))
=> '#hello-world!
```

Can you write an *append-streams** function which is like *append-streams*, but appends a variable number of streams? What should (**append-streams***) reduce to? (Q17)

Some of the stream functions can be applied to infinite streams safely and some can not. *Stream->list* is a function that accepts only finite streams:

```
(stream->list (stream 'foo id all id none :f)) => bottom
```

Which of the functions presented above are safe for infinite streams? (Q18)

Invent some stream functions of your own. Are there any list operations that cannot be expressed using streams?

8.3 ml-style records

A *record* is a set of ordered tuples similar to an association list. The difference between a record and an alist is that a record has a fixed number of members. Adding or removing members changes the *type* of the record. The following structure resembles a record:

```
((food ginger) (type root) (vegetarian :t))
```

Each sublist of the record is called a *field*. The car part of each field contains the *tag* of this field and its cadr part contains a value associated with that tag.

The *ML* language¹⁴ provides a highly flexible mechanism for creating and manipulating records, which will be emulated by the following code as far as this is possible in a dynamically typed, purely functional environment.

Records make use of numbers, so any of the math packages has to be loaded. When none of them has been loaded before, **rmath** is loaded by default. This solution allows to use records in combination with any of the math packages:

¹⁴ ML ("Meta Language") is a statically typed functional programming language invented by Robin Milner et al. at the University of Edinburgh in 1973. It was probably the first language to employ "type inference". See also: <http://www.smlnj.org>.

zen style programming

```
(or (defined 'nmath)
    (defined 'imath)
    (defined 'rmath)
    (load ~rmath))
```

In order to distinguish records from other data types, a unique instance [page 55] of the datum `'(%record)` is created. This datum will be used to tag records.

```
(define record-tag (list '%record))
```

The below procedures determine the types of given forms. Note that both closures and records are lists at the same time. There is no way to implement more strict type checking in `zenlisp`.

```
(define (pair-p x) (not (atom x)))

(define (boolean-p x)
  (or (eq x :t)
      (eq x :f)))

(define (closure-p x)
  (and (pair-p x)
       (eq (car x) 'closure)))

(define (record-p x)
  (and (pair-p x)
       (eq (car x) record-tag)))
```

List->record turns a list of two-element lists (tag/value tuples) into a record:

```
(list->record '((food marmelade) (type processed)))
=> ((%record) (food marmelade) (type processed))
(list->record '(foo bar))
=> bottom
```

It also checks whether the pairs have the appropriate format, but it does not check for duplicate tags. Feel free to improve the code on your own.

```
(define (list->record a)
  (letrec
    ((valid-fields-p
      (lambda (a)
        (or (null a)
            (and (pair-p (car a))
                 (atom (caar a))
                 (pair-p (cдар a))
                 (null (cddar a))
                 (valid-fields-p (cdr a))))))
    (cond ((valid-fields-p a) (cons record-tag a))
          (t (bottom 'bad-record-structure a)))))
```

The *record* function is the principal record constructor. It assembles a record from a set of tag/value tuples:

100

```
(record '(foo bar) '(baz goo)) => ((%record) (foo bar) (baz goo))
```

It is based on *list->record*, but evaluates its individual arguments before constructing the record:

```
(define (record . x) (list->record x))
```

Note that you *cannot* create record literals like `'((%record) (foo bar))`, because the `(%record)` part of that structure is not identical to the unique instance bound to *record-tag*:

```
(record-p (record '(foo bar))) => :t  
(record-p '(%record) (foo bar))) => :f
```

Record->list is the reverse operation of *list->record*.

```
(define (record->list r)  
  (cond ((record-p r) (cdr r))  
        (t (bottom 'expected-record-got r))))
```

The *record-field* function extracts the field with a given tag from a record and *record-ref* extracts the value associated with a given tag:

```
(record-field (record '(a #1) '(b #2)) 'b) => '(b #2)  
(record-ref (record '(a #1) '(b #2)) 'b) => #2
```

Both of them reduce to bottom when either their first argument is not a record or the second argument does not occur as a tag in the given record.

```
(define (record-field r tag)  
  (let ((v (assq tag (record->list r))))  
    (cond (v v)  
          (t (bottom 'no-such-tag  
                    (list 'record: r 'tag: tag))))))  
  
(define (record-ref r tag) (cadr (record-field r tag)))
```

Type-of returns a symbol that indicates the type of a given form. Note that the list does not occur in *type-of*. It returns `'pair` for pairs (and hence also for lists) and even for empty lists.

```
(define (type-of x)  
  (cond ((boolean-p x) 'boolean)  
        ((null x) 'pair)  
        ((atom x) 'symbol)  
        ((number-p x) 'number)  
        ((record-p x) 'record)  
        ((closure-p x) 'function)  
        ((pair-p x) 'pair)  
        (t (bottom 'unknown-type x))))
```

Two records are *equal*, if

- they have the same number of fields;
- their fields have the same tags;
- fields with identical tags have equal values.

zen style programming

The fields of two equal records need not appear in the same order. For example, the following two records are equal:

```
(record '(foo #1) '(bar #2))  
(record '(bar #2) '(foo #1))
```

If records contain records, embedded records are compared recursively. The *record-equal* predicate compares records:

```
(define (record-equal r1 r2)  
  (letrec  
    ((equal-fields-p  
      (lambda (r1 r2)  
        (cond ((null r1) :t)  
              (t (let ((x (assq (caar r1) r2)))  
                   (and x  
                        (equal (cadar r1) (cadr x))  
                        (equal-fields-p (cdr r1) r2)))))))  
    (let ((lr1 (record->list r1))  
          (lr2 (record->list r2)))  
      (and (= (length lr1) (length lr2))  
           (equal-fields-p lr1 lr2)))))
```

The **equal** predicate is extended in order to handle records, too:

```
(define (equal a b)  
  (cond ((eq a b) :t)  
        ((and (pair-p a) (pair-p b))  
         (and (equal (car a) (car b))  
              (equal (cdr a) (cdr b))))  
        ((record-p a)  
         (and (record-p b)  
              (record-equal a b)))  
        (t :f)))
```

The *signature* of a record **r** is another record that contains the same tags, but instead of the values of **r** it contains the *types* of its values, e.g.:

```
(record-signature (record '(food apple) '(weight #550) '(vegetarian :t)))  
=> '((%record) (food symbol) (weight number) (vegetarian boolean))
```

A record containing embedded records has a recursive signature:

```
(record-signature (record (list 'p1 (record '(x #0) '(y #0)))  
                          (list 'p2 (record '(dx #0) '(dy #0)))))  
=> '((%record) (p1 (record ((%record) (x number) (y number))))  
      (p2 (record ((%record) (dx number) (dy number)))))
```

The *record-signature* function creates the signature of a record:

```
(define (record-signature r)  
  (letrec
```

```
((make-sig
  (lambda (x)
    (map (lambda (x)
      (cond ((record-p (cadr x))
        (list (car x)
          (list (type-of (cadr x))
            (record-signature (cadr x)))))
        (t (list (car x) (type-of (cadr x)))))
      x))))
  (list->record (make-sig (record->list r)))))
```

The *record-set* function creates a fresh record in which the value of the given field is replaced with a new value:

```
(define r (record '(food cucumber)))
r => '(%record) (food cucumber))

(record-set r 'food 'melon) => '(%record) (food melon))
r => '(%record) (food cucumber))
```

Note that *record-set* does not alter the original record. When the given tag does not occur in the given record or the value associated with the tag has a different type than the new value, *record-set* yields bottom:

```
(record-set (record '(food salt)) 'zzz 'baz) => bottom ; unknown tag

(record-set (record '(food salt)) 'food :f) => bottom ; type mismatch
```

When replacing values of the type record, the old and the new value must have the same signature:

```
(define r (record (list 'menu (record '(food apple)))))

(record-set r 'food (record '(food (x y z))))
=> bottom ; type mismatch, expected: (%record) (food symbol))
      got: (%record) (food pair))

(record-set r 'menu (record '(food orange)))
=> '(%record) (menu (%record) (food orange)))
```

Here is the code of the *record-set* function.

```
(define (record-set r tag v)
  (letrec
    ((subst
      (lambda (r old new)
        (cond ((null r) ())
              ((eq old (car r))
               (cons new (cdr r)))
              (t (cons (car r)
                (subst (cdr r) old new))))))
    (subst (cdr r) old new))))
```

zen style programming

```
(type-mismatch
  (lambda ()
    (bottom 'type-mismatch
      (list 'record: r 'tag: tag 'value: v))))
(let ((f (record-field r tag)))
  (let ((b (cdr f)))
    (cond ((eq (type-of (car b)) (type-of v))
      (cond ((or (not (record-p v))
        (record-equal
          (record-signature (car b))
          (record-signature v)))
        (subst r f (list (car f) v)))
      (t (type-mismatch))))
    (t (type-mismatch))))))
```

Because *record-set* only replaces values of matching types, it makes records as type-safe as ML records. Note that there is no need to declare record types in order to achieve type safety. The type of a record is determined by extracting its signature.

Explicit type checking or dispatch can be added to a function by means of the *assert-record-type* and *record-type-matches-p* functions.

Record-type-matches-p is a predicate that returns truth if a given record matches a given signature. It is merely an abbreviation:

```
(define (record-type-matches-p sig r)
  (record-equal sig (record-signature r)))
```

It is used as follows:

```
(define point-type (record-signature (record '(x #0) '(y #0))))
...
(define (some-function x)
  (cond ((record-type-matches-p point-type x)
    code handling point records...)
    (t code handling other types ...)))
```

Assert-record-type is similar, but used to *make sure* that an argument has a given type:

```
(define (some-function r)
  (function-expecting-a-point (assert-record-type point-type r)))
```

As long as a record passed to *assert-record-type* has the expected signature, the function simply returns the record. When its type does not match, it aborts the computation and reduces to bottom. Here is the code of *assert-record-type*:

```
(define (assert-record-type sig r)
  (cond ((not (record-type-matches-p sig r))
    (bottom 'record-type-assertion-failed
      (list 'signature: sig 'record: r)))
    (t r)))
```

What signature does the signature of a record have? And the signature of the signature of a record? Give some examples. **(Q19)**

Of course, the record type is most useful in languages that support mutation of values, so this implementation is merely a proof of concept. Can you imagine any real use for the code of this section?

Which modification(s) would you apply to the code when porting it to a language supporting mutable values (like Scheme)?

9. compilers

9.1 translating infix to prefix

The *infix->prefix* function presented in this section implements a so-called *recursive descent parser*. A *parser* is a program that analyses the syntactical structure of its input and transforms it into some other representation. In the example presented in this section, the “other representation” is that of *zenlisp* expressions. *Recursive descent* is a parsing technique. It will be explained in detail in this section.

The first question when designing a parser is how to represent its input. In the real world, this would most probably be a string or a “text file”, but because *zenlisp* does not provide either of them, lists will be used instead. For example, the formula

$$x^2 + y$$

would be written as

```
'#x^2+y
```

The *infix->prefix* program will analyze formulae of the above form and translate them to corresponding *zenlisp* forms, e.g.:

```
(infix->prefix '#x^2+y)    => '(+ (expt x '#2) y)
(infix->prefix '#x*2+y*3)  => '(+ (* x '#2) (* y '#3))
(infix->prefix '#x*[2+y]*3) => '(* (* x (+ '#2 y)) '#3)
```

The parser will recognize the following symbols in its input:

Input	Meaning
[a-z]	symbol (single letters only)
[0-9] +	integer numbers (sequences of digits)
+	addition
-	subtraction or negation (depends on context)
*	multiplication
/	division
^	exponentiation
[]	grouping of subexpressions

In order to describe the input of a parser in detail, though, a little digression is in order.

9.1.1 formal grammars

Technically speaking, the input of the *infix->prefix* program will be a *formal language*. The

symbols given at the end of the previous section are the *lexemes* of that language. Any non-empty sequence of such lexemes is a *sentence* of that language (although not necessarily a *well-formed* one). The following sequences are sentences:

```
aaaa
a+b
a+-b^
[x]y
]]]
```

Just like natural languages formal languages have grammars that are used to construct *well-formed sentences*. A well-formed sentence of a formal language is also called a *program* of that language. Intuition may tell you that the following sentences are programs:

```
a+b
x*y+z
x-17
```

But what about these:

```
xyz
a--b
p[q]
```

Intuition is fine but hard to implement in a parser, so we need some means of describing a grammar *formally*. This is where *BNF* (“Backus Normal Form” or “Backus Naur Form”) comes into play. BNF is a notation for describing the grammars of programming languages formally. The basic building stone of BNF descriptions is the *production*. Productions look like this:

```
<sum> := symbol '+' symbol
        | symbol '-' symbol
```

The “:=” operator reads “is defined as” or “can be written as”. The “|” denotes a logical or. So the above production says: “a <sum> can be written as a `symbol` followed by + and another `symbol` or as a `symbol` followed by - and another `symbol`”.

Each name that stands alone denotes a lexeme, which is also called a *terminal symbol* in compiler speak. Names like `symbol` typically represent classes of symbols. In the language we are about to define, `symbol` would denote the class containing the lexemes

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

and `number` would represent the (infinite) class containing the lexemes

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 ...
```

A name that is enclosed in apostrophes is a terminal symbol that represents itself, so `'+'` represents the lexeme + and `'['` represents the lexeme [.

Names enclosed in angle brackets, like `<sum>`, are so-called *non-terminal symbols*. They represent

zen style programming

productions. The conventions used for terminals, non-terminals and even the operators differ between textbooks, but the fundamental principles are always the same: the lefthand side of a production gives a name to the production and the righthand side describes what the lefthand side can be replaced with. For example, according to the above production, these are `<sum>`s (here the generic arrow means “according to the rule”):

```
a+b  →  symbol '+' symbol
x-y  →  symbol '-' symbol
```

Any sentence not matching the rules of `<sum>` is not a valid program of `<sum>`. The term *rule* is sometimes used as a synonym of “production”. In this text, though, “rule” will be used to refer to one alternative of a production, so the `<sum>` production has two rules. These rules could also be written as separate productions:

```
<sum> := symbol '+' symbol
<sum> := symbol '-' symbol
```

This is rarely done, though, because the “or” operator makes productions more readable.

The omnipresent principle of recursion also plays a central role in BNF productions. It is used to describe sentences of variable length:

```
<a*> := 'a'
      | 'a' <a*>
```

The production `<a*>` matches any positive number of `a`s. Its rules say that `<a*>` may be replaced by a single `a` or by an `a` followed by another `<a*>` (which in turn may be either a single `a` or an `a` followed by another `<a*>` (which ... you get the idea)), so these rules “produce”¹⁵ the sentences

```
a      →  'a'
aa     →  'a' <a*> 'a'
aaa    →  'a' <a*> 'a' <a*> 'a'
aaaa   →  'a' <a*> 'a' <a*> 'a' <a*> 'a'
aaaaa  →  'a' <a*> 'a' <a*> 'a' <a*> 'a' <a*> 'a'
...
```

To say that a “production produces” a set of sentences implies that the production matches these sentences. The set of sentences produced by a production is exactly the set of programs accepted by that production.

The principle of recursion can be used, for instance, to form `<sum>`’s with any number of operands:

```
<sum> := symbol
      | symbol '+' <sum>
      | symbol '-' <sum>
```

¹⁵ Yes, this is why a set of rules is called a “production”.

Here are some of the sentences produced by this version of `<sum>`:

```
x      →  symbol
x+y    →  symbol '+' <sum> symbol
x+y-z  →  symbol '+' <sum> symbol '-' <sum> symbol
```

The above production can (almost) be used to describe a part of the language accepted by the *infix->prefix* parser. However, the parser will accept not just symbols as operands but also numbers, and it will accept terms and exponents, too. One production is not enough to cover all of these, so multiple productions will be combined to form a *grammar*. The following grammar accepts a language with both numbers and symbols as operands in sums:

```
<sum>  := <factor>
        | <factor> '+' <sum>
        | <factor> '-' <sum>

<factor> := symbol
          | number
```

In real-world math formulae operations like multiplication and division “bind stronger” than, for example, addition and subtraction. A compiler writer would say that multiplication and division have a “higher precedence” than addition and subtraction. Precedence is easy to implement in grammars (see the above grammar for the definition of `<factor>`):

```
<term> := <factor>
        | <factor> '*' <term>

<sum>  := <term>
        | <term> '+' <sum>
```

Here `<term>` works in the same way as `<sum>` in the previous grammar. `<Sum>`s are now composed of `<term>`s. Because a complete `<term>` has to be parsed before a `<sum>` can be produced, `<term>`s bind stronger than `<sum>`s. In other words: a `<sum>` is a `<term>` followed by optional `<sum>` operations.

Let us check some intuitively well-formed sentences against this grammar:

```
x      →  <sum> <term> <factor> 'x'
x+y    →  <sum> <term> <factor> 'x' '+' <sum> <term> <factor> 'y'
x+y*z  →  <sum> <term> <factor> 'x' '+' <sum> <term> <factor> 'y'
        '*' <term> <factor> 'z'
x*y+z  →  <sum> <term> <factor> 'x' '*' <term> <factor> 'y'
        '+' <sum> <term> <factor> 'z'
```

Okay, this is the point where things become a bit messy, because the linear representation is not really suitable for representing sentences. This is why the output of parsers is typically presented in tree form.

The tree in figure 5 shows the *syntax tree* of the formula $x+y*z$. A syntax tree is sometimes also

zen style programming

called a *parse tree*. The square boxes represent non-terminals, the circles terminals. Following the outer edge of the tree visits the terminals in the order in which they appear in the original formula.

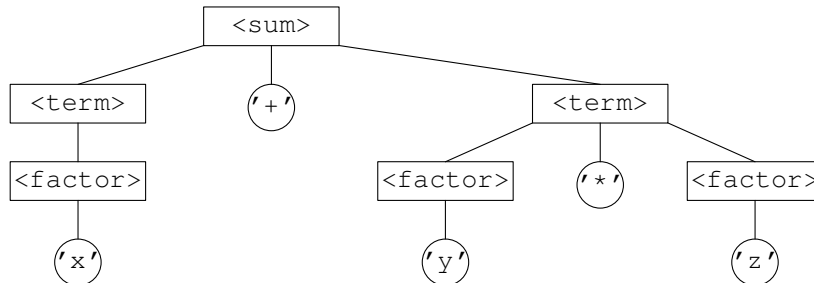


Fig. 5 – syntax tree of $x+y*z$

Because a `<term>` can be part of a `<sum>`, but a `<sum>` can never be part of a `<term>`, `<term>`s are always contained in `<sum>` trees and therefore, term operations are visited before sum operations when traversing the tree using “depth-first” traversal. As a consequence of this order, term operations have a higher precedence than sum operations.

Depth-first traversal of a tree means that subtrees are always visited before processing parents. For instance, a tree can be converted to *reverse polish notation* (*suffix notation*) by visiting the left branch of each non-terminal node first, then visiting the right branch and finally emitting the terminal attached to the node (if any). Traversing the above tree would yield:

`x y z * +`

Emitting the operand *before* descending into branches would yield prefix notation:

`+ x * y z`

Adding parentheses gives a zenlisp program:

`(+ x (* y z))`

Note that the precedence of the sum and term operators is preserved in all notations: tree, suffix, and prefix.

We now know how to define grammars, represent parsed text, and even how to generate zenlisp programs. What is missing is the full grammar of the input language. Here it is:

```
<sum> := <term>
      | <term> '+' <sum>
      | <term> '-' <sum>

<term> := <power>
      | <power> '**' <term>
      | <power> <term>
      | <power> '/' <term>
```

```
<power> := <factor>
        | <factor> '^' <power>

<factor> := symbol
        | number
        | '-' <factor>
        | '[' <sum> ']'
```

Each well-formed sentence accepted by the *infix-prefix* parser is a production of `<sum>`. Note that the rule

```
<term> := <power> <term>
```

allows to abbreviate $x*y$ as xy . Of course, this is only possible because variables are single-letter lexemes. The rule

```
<factor> := '[' <sum> ']'
```

gives a `<sum>` (no matter which operators it eventually contains) the precedence of a `<factor>`, thereby allowing to group subexpressions just like in math formulae. Also note that the minus prefix (which negates a factor) has the highest possible precedence, so

$-x^2$

actually means

$(-x)^2$

Now that the grammar for the *infix->prefix* parser has been specified formally, there is no need to rely on intuition any longer, and the implementation of the parser is quite straight-forward.

Well, almost straight-forward. There is one subtle detail left to discuss.

9.1.2 left versus right recursion

The recursive productions used in the grammars shown so far are so-called *right-recursive* productions. They are termed so because the recursive rules of each production recurse at the rightmost end, like this:

```
<diff> := <factor>
        | <factor> '-' <diff>
```

A right-recursive parser for this small grammar can be easily packaged in a function (although some of the details will be explained later and are left to the imagination of the reader for now):

```
(define (diff x)
  (let ((left (factor x)))
    (cond ((null (rest left)) left)
```

zen style programming

```
((eq (car-of-rest left) '-)
  (let ((right (diff (cdr-of-rest left))))
    (list '- (expr left) (expr right))))
(t left)))
```

This parser recognizes chains of “-” operators just fine, but the parse tree generated by it associates operators to the right, as shown in figure 6.

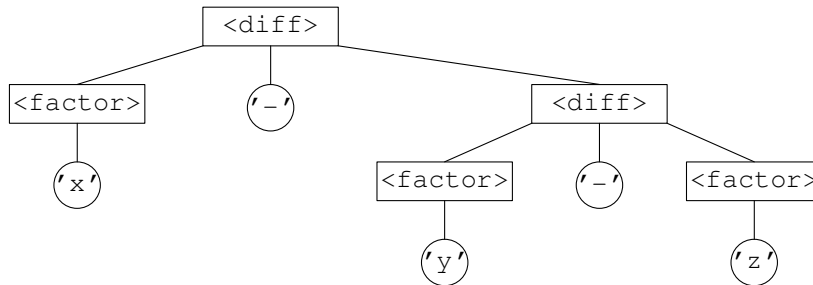


Fig. 6 – right-associative syntax tree of $x-y-z$

When traversing this tree, the resulting prefix expression would be

```
(- x (- y z))
```

which in turn translates back to

$x-(y-z)$

It clearly *should* give $(x-y)-z$, though. In other words, the parser gives the “-” operator the wrong *associativity*:

```
x-y-z = (- x (- y z)) ; right-associative
x-y-z = (- (- x y) z) ; left-associative
```

Left-associativity is needed, but right-associativity is delivered. In a grammar this is easy to fix by simply rewriting it in a *left-recursive* way, where recursion occurs at the beginning of each rule:

```
<diff> := <factor>
        | <diff> '-' <factor>
```

Unfortunately, this approach cannot be implemented in the way outlined above, as can be seen in the following code fragment:

```
(define (diff x)
  (let ((left (diff x)))
    (cond ((null (rest left)) left)
          ...)))
```

Because this hypothetical parser function would recurse immediately, it would never reach its trivial case and hence recurse indefinitely.

By rewriting the function slightly, left recursion can be eliminated, though:

```
(define (diff2 out in)
  (cond ((null in)
        out)
        ((eq (car in) '-')
         (let ((right (factor (cdr in))))
           (diff2 (list '- out (expr right))
                  (rest right))))
        (t out)))

(define (diff x)
  (let ((left (factor x)))
    (diff2 (expr left) (rest left))))
```

This version of the *diff* function passes the first factor to *diff2* which then collects “-” operators (if any). *Diff2* descends immediately into *factor* rather than into itself and *then* recurses to collect more factors. To prove that it actually builds a left-associative expressions is left as an exercise to the reader.

9.1.2 implementation

The parser implemented in this section is a so-called *recursive descent parser*. It implements each production of a formal grammar in a separate function (e.g. the *factor* function implements the rules of <factor>, etc). Like productions of a grammar, these functions form a hierarchy. Lexemes are matched by *descending* into this hierarchy until a function is found that recognizes the given lexeme. The method is called *recursive descent*, because functions recurse to accept variable-length input or to ascend back to higher levels of the hierarchy (e.g. for handling parenthesized subexpressions).

Here comes the code:

```
(require '~rmath)

(define (infix->prefix x)
  (letrec
```

Check whether *x* is a symbol:

```
((symbol-p
  (lambda (x)
    (and (memq x '#abcdefghijklmnopqrstuvwxyz) :t)))
```

Collect a numeric literal from the input and return a list containing the literal and the rest of the input, e.g.:

```
(number '#123+x ()) => '('#123 #+x)
```

(The result really does contain a quoted number, this is not a mistake.)

zen style programming

```
(number
  (lambda (x r)
    (cond ((or (null x)
               (not (digitp (car x))))
          (list (list 'quote (reverse r)) x))
          (t (number (cdr x) (cons (car x) r))))))
```

Extract a symbol from the input (analogous to *number*).

```
(symbol
  (lambda (x)
    (list (car x) (cdr x))))
```

The following convenience functions are used to access the individual parts of partially translated formulae. For example, the *number* function may return the value `' ('#1 #-x*7)`. The *car* part of such a result is a *zenlisp* expression, which is extracted using *expr*:

```
(expr ' ('#1 #-x*7)) => '#1
```

Rest extracts the remaining input that still is to be parsed, *car-of-rest* the first character of the rest, and *cdr-of-rest* the rest with its first character removed:

```
(rest ' ('#1 #-x*7)) => '#-x*7
(car-of-rest ' ('#1 #-x*7)) => '-'
(cdr-of-rest ' ('#1 #-x*7)) => '#x*7
```

While parsing a formula, the intermediate steps are always kept in *expr/rest* tuples.

```
(expr car)
(rest cadr)
(car-of-rest caadr)
(cdr-of-rest cdadr)
```

The *factor* function parses a factor as described by the `<factor>` production. Like all parsing functions, it returns an *expr/rest* tuple:

```
(factor '#-123+456) => '((- '#123) '#+456)
```

Factor, being at the bottom of the recursive descent chain, cannot accept empty input. Hence it aborts with a “syntax error” when an empty list is passed to it. It also reports a syntax error when the lexeme at the beginning of its input does not match any of its rules, e.g.:

```
(factor '#+++ ) => bottom
```

When it finds an opening parenthesis, it makes sure that a closing parenthesis follows after the `<sum>` between the parentheses.

```
; <factor> := '[' <sum> ']'
;          | '-' <factor>
;          | number
;          | symbol
```

```
(factor
  (lambda (x)
    (cond ((null x)
           (bottom 'syntax 'error 'at: x))
          ((eq (car x) '[])
           (let ((xsub (sum (cdr x))))
             (cond ((null (rest xsub))
                    (bottom 'missing-right-paren))
                   ((eq (car-of-rest xsub) '[])
                    (list (expr xsub) (cdr-of-rest xsub)))
                   (t (bottom 'missing-right-paren))))))
          ((eq (car x) '-')
           (let ((fac (factor (cdr x))))
             (list (list '- (expr fac)) (rest fac))))
          ((digitp (car x))
           (number x ()))
          ((symbol-p (car x))
           (symbol x))
          (t (bottom 'syntax 'error 'at: x))))))
```

Power implements the <power> production. It always parses one factor (which is why it does not accept empty input) and then recurses to collect more factors connected by ^ operators to the expression parsed so far:

```
(power '#x^y^z+5) => '((expt x (expt y z)) #+5)
```

It stops parsing when a factor is followed by something that is not a ^ operator (or when the input is exhausted).

Note that right recursion is desired in *power* because powers actually do associate to the right.

```
; <power> := <factor>
;         | <factor> ^ <power>
(power (lambda (x)
  (let ((left (factor x)))
    (cond ((null (rest left)) left)
          ((eq (car-of-rest left) '^)
           (let ((right (power (cdr-of-rest left))))
             (list (list 'expt (expr left) (expr right))
                   (rest right))))
          (t left))))))
```

Term is similar to *power* but applies the left recursion hack described in the previous subsection. It accepts the * and / operators instead of ^. It also accepts xx as an abbreviation for x*x:

```
; term := power
;      | power Symbol
;      | power * term
;      | power / term
```

zen style programming

```
(term2
  (lambda (out in)
    (cond ((null in) (list out in))
          ((symbol-p (car in))
           (let ((right (power in)))
             (term2 (list '* out (expr right))
                     (rest right))))
          ((eq (car in) '*)
           (let ((right (power (cdr in))))
             (term2 (list '* out (expr right))
                     (rest right))))
          ((eq (car in) '/')
           (let ((right (power (cdr in))))
             (term2 (list '/ out (expr right))
                     (rest right))))
          (t (list out in)))))
(term
  (lambda (x)
    (let ((left (power x)))
      (term2 (expr left) (rest left)))))
```

All of the parsing functions follow the same scheme, so `sum` is basically like `factor`, `power`, and `term`. It differs from them only in the operators being accepted. In a later chapter, a more general approach will be shown that avoids this code duplication.

```
; sum := term
;      | term + sum
;      | term - sum
(sum2
  (lambda (out in)
    (cond ((null in) (list out in))
          ((eq (car in) '+)
           (let ((right (term (cdr in))))
             (sum2 (list '+ out (expr right))
                   (rest right))))
          ((eq (car in) '-')
           (let ((right (term (cdr in))))
             (sum2 (list '- out (expr right))
                   (rest right))))
          (t (list out in)))))
(sum
  (lambda (x)
    (let ((left (term x)))
      (sum2 (expr left) (rest left)))))
```

The body of *infix->prefix* passes its argument to `sum`. When the code in that argument could be parsed successfully, the rest part of the resulting tuple will be empty. Otherwise none of the functions in the recursive descent chain had a rule for handling the lexeme at the beginning of the rest, so a non-empty rest indicates a syntax error. E.g.:

```
(sum 'x+y@z) => '((+ x y) #@z)
```

When the rest part of the tuple is empty, the body simple returns the expression part, which contains the complete prefix expression at this point.

```
(let ((px (sum x)))
  (cond ((not (null (rest px)))
        (bottom (list 'syntax 'error 'at: (cadr px))))
        (t (expr px)))))
```

The *infix->prefix* program does not explicitly create a syntax tree. Nevertheless it uses the approach described in this section to convert infix to prefix notation. How does it do this? What are the inner nodes of the tree? (Q20)

Some people would argue that making $-x^2$ equal to $(-x)^2$ is a bad idea. Can you change the precedence of the unary minus operator in such a way that it still binds stronger than the term operators $*$ and $/$ but not as strong as exponentiation ($^$)? Implement your modification as a BNF grammar as well as in *zenlisp* code. (Q21)

9.2 translating prefix to infix

As its name suggests, *prefix->infix* is the inverse function of *infix->prefix*. It takes a prefix expression represented by a *zenlisp* form and turns it into an infix expression. Like its cousin, it preserves precedence and associativity. It adds parentheses where necessary:

```
(prefix->infix '(+ (expt x '#2) y))      => '#x^2+y
(prefix->infix '(+ (* x '#2) (* y '#3)))  => '#x*2+y*3
(prefix->infix '(- (- a b) (- c d)))      => '#a-b-[c-d]
```

A combination of *infix->prefix* and *prefix->infix* can be used to remove (most) superfluous parentheses from a formula:

```
(prefix->infix (infix->prefix '#[a+b]-[c+d])) => '#a+b-[c+d]
```

There does not appear to be a commonly used named for functions like *prefix->infix*. What they do is a mixture of tree traversal and code synthesis for a virtual “infix machine”. The largest part of the program deals with the generation of parentheses.

Here is the code:

```
(define (prefix->infix x)
  (letrec
```

The *ops* alist maps *zenlisp* functions to infix operators, *left* contains all functions that map to left-associative operators, and *precedence* contains operators in groups of descending precedence.

```
((ops '((+ . +) (- . -) (* . *) (/ . /)) (expt . ^)))
(left '#+*-/))
(precedence '(high ([]) (expt) (* /) (+ -) low))
```

zen style programming

The following predicates are used to check properties of forms. For instance, *function-p* checks whether a form denotes a function and *left-assoc-p* evaluates to truth when its argument is a function resembling a left-associative operator. These functions should be pretty much self-explanatory:

```
(function-p
  (lambda (x)
    (and (memq x '(+ - * / expt)) :t)))
(left-assoc-p
  (lambda (x)
    (and (memq x left))))
(symbol-p
  (lambda (x)
    (and (memq x '#abcdefghijklmnopqrstuvwxyz) :t)))
(numeric-p
  (lambda (x)
    (and (not (atom x))
          (eq (car x) 'quote))))
(atomic-p
  (lambda (x)
    (or (function-p x)
        (symbol-p x)
        (numeric-p x))))
```

Unary-p checks whether a form represents a unary function application:

```
(unary-p
  (lambda (x)
    (and (not (null (cdr x)))
          (null (cddr x)))))
```

The *higher-prec-p* function finds out whether the formula *x* has a higher precedence than the formula *y*. For instance:

```
(higher-prec-p '#1 '(+ a b))      => :t
(higher-prec-p '(* a b) (+ a b)) => :t
(higher-prec-p '(- a) (expt a b)) => :t
(higher-prec-p '(- a) 'a)         => :f
```

Atomic forms (symbols, numbers) have the highest precedence (*because* they are atomic), followed by unary operators and the precedence rules expressed in the *precedence* list.

```
(higher-prec-p
  (lambda (x y)
    (letrec
      ((hpp (lambda (x y prec)
               (cond ((atom prec) :f)
                     ((memq x (car prec))
                      (not (memq y (car prec))))
                     ((memq y (car prec)) :f)
                     (t (hpp x y (cdr prec)))))))
```

```
(cond ((atomic-p x) (not (atomic-p y)))
      ((atomic-p y) :f)
      ((unary-p x) (not (unary-p y)))
      ((unary-p y) :f)
      (t (hpp (car x) (car y) (cdr precedence))))))
```

Paren places parentheses ([and]) around the given expression, but never around atoms.

```
(paren
  (lambda (x)
    (cond ((atomic-p x) x)
          (t (list '[' x])))))
```

The *add-parens* function adds parenthesis tags to an expression:

```
(add-parens '(* (+ a b) c)) => '(* ([' (+ a b)] c)
```

The [] (“parens”) symbol indicates that the following subexpression must be put in parentheses to preserve precedence when converting it to infix. *Add-parens* tags only those subexpressions that really need explicit grouping:

```
(add-parens '(+ (* a b) c)) => '(+ (* a b) c)
```

When an atomic form is passed to *add-parens*, it simply returns it, otherwise it first processes subforms by recursing through **map**. Finally it applies the precedence rules we know.

When the current formula is the application of a unary function and the argument is neither atomic nor another unary function, the argument is put in parentheses.

When the current formula is the application of a left-associative binary function, parentheses are placed around the left argument if the operator of the formula has a higher precedence than that of the argument:

```
(add-parens '(* (+ a b) c)) => '(* ([' (+ a b)] c)
```

The second argument is put in parentheses if it has not a higher precedence than the operator of the formula (that is, if its precedence is lower than or equal to the operator of the formula):

```
(add-parens '(* a (+ b c))) => '(* a ([' (+ b c)]))
```

The latter rule also tags operations that are grouped to the right at the same precedence level.

```
(add-parens '(- a (- b c))) => '(- a ([' (- b c)]))
```

The rules for right-associative operations (the catch-all clause of the inner **cond** of *add-parens*) are similar to the above, but the rules are changed in such a way that operations that group to the left at the same precedence level are tagged.

```
(add-parens
  (lambda (x)
    (cond
```

zen style programming

```
((atomic-p x) x)
(t (let ((x (map add-parens x)))
  (cond ((unary-p x)
    (cond ((atomic-p (cadr x)) x)
      ((unary-p (cadr x)) x)
      (t (list (car x)
        (paren (cadr x))))))
    ((left-assoc-p (car x))
      (list (car x)
        (cond ((higher-prec-p x (cadr x))
          (paren (cadr x)))
          (t (cadr x)))
        (cond ((higher-prec-p (caddr x) x)
          (caddr x))
          (t (paren (caddr x))))))
    (t (list (car x)
      (cond ((higher-prec-p (cadr x) x)
        (cadr x))
        (t (paren (cadr x))))
      (cond ((higher-prec-p x (caddr x))
        (paren (caddr x)))
        (t (caddr x))))))))))
```

The *infix* function traverses the tree represented by a zenlisp form and emits an infix expression. It puts parentheses around subexpressions tagged by []:

```
(infix '(* (+ x y) z))      => '#x+y*z
(infix '(* ([ (+ x y)) z)) => '#[x+y]*z
```

Infix also checks the consistency (syntax) of the zenlisp expression and report errors.

```
(infix
  (lambda (x)
    (cond
      ((numeric-p x)
        (cadr x))
      ((symbol-p x)
        (list x))
      ((and (eq (car x) '-')
        (not (atom (cdr x)))
        (null (caddr x)))
        (append '#- (infix (cadr x))))
      ((and (eq (car x) '[')
        (not (atom (cdr x)))
        (null (caddr x)))
        (append '#[ (infix (cadr x)) '#]))
      ((and (not (atom x))
        (not (atom (cdr x)))
        (not (atom (caddr x)))
        (null (caddr x))
        (function-p (car x)))
```

```
(append (infix (cadr x))
        (list (cdr (assq (car x) ops))
              (infix (caddr x))))
(t (bottom (list 'syntax 'error: x))))))
```

The main body simply combines *add-parens* and *infix*.

```
(infix (add-parens x)))
```

Can you rewrite *prefix->infix* in such a way that it puts parentheses around *all* operations, thereby making precedence explicit? What practical applications could such a transformation have? (Q22)

Can you make *prefix->infix* emit reverse polish notation (RPN, suffix notation) instead of infix notation? E.g.:

```
(prefix->rpn '(* (+ x y) z)) => '#xy+z*
```

Does RPN ever need parentheses? Why? (Q23)

Note that *prefix->infix* sometimes adds superfluous parentheses:

```
(prefix->infix '(+ x (+ y z))) => '#x+[y+z]
```

This is because the program does not implement *commutativity*. An operator **o** is commutative, if chains of **o** operations are independent of associativity:

$$(a \mathbf{o} b) \mathbf{o} c = a \mathbf{o} (b \mathbf{o} c)$$

The + and * operators are commutative. Can you implement rules in *add-parens* that recognize commutativity and skip parentheses where possible?

9.3 regular expressions

A *regular expression* (RE) is a pattern that is used to match sequences of characters. An RE is more general than a string, because it allows to include “special” characters which match classes or characters or sequences of characters. The details will be explained immediately.

The functions introduced in this chapter implement what is now called “basic” regular expressions. This is the RE format used in “traditional” Unix and early versions of the `grep(1)` utility.

A regular expression consists of a set of characters and operators. Most characters simply match themselves, so the RE `foo` would match the sequence `foo`, or even the sequence `afoob` because it contains `foo`.

The following characters have special meanings in REs: ¹⁶

¹⁶ Unix-style REs would use the dot (.) instead of the underscore (_), but this cannot be done in `zenlisp` because the dot is reserved for dotted pairs.

zen style programming

`_ [] ^ $ * + ? \`

When one of these characters is to be matched literally, it must be prefixed with a backslash (`\`). Otherwise it is interpreted as an operator. The meanings of the operators are as follows:

<code>[c₁...]</code>	A <i>character class</i> matches any character contained in the square brackets.
<code>[^c₁...]</code>	When the first character between the brackets is <code>^</code> , the class matches any character <i>not</i> contained in it.
<code>[c₁-c₂ ...]</code>	When a minus sign occurs in a class, it is replaced with the characters that occur between the character in front of the minus sign and the character following the sign, e.g.: <code>[0-9]</code> expands to <code>[0123456789]</code> .
<code>_</code>	This is the class containing <i>all</i> characters, so it matches any character.
<code>^</code>	Matches the beginning of a sequence.
<code>\$</code>	Matches the end of a sequence.
<code>*</code>	Matches <i>zero or more</i> occurrences of the preceding character or class.
<code>+</code>	Matches <i>at least one</i> occurrence of the preceding character or class.
<code>?</code>	Matches <i>zero or one</i> occurrence of the preceding character or class.
<code>\c</code>	Matches the character <code>c</code> literally, even if it is an operator.

Here are some sample REs:

<code>[A-Za-z]+</code>	matches any alphabetic sequence
<code>[A-Za-z]+[0-9]*</code>	matches any alphabetic sequence followed by an optional numeric sequence
<code>[a-z][0-9]?</code>	matches any lower-case letter followed by an optional digit
<code>_*</code>	matches any sequence of any length (even empty ones)
<code>_+</code>	matches any sequence of any length (but not empty ones)
<code>**</code>	matches any sequence of asterisks

The following code contains two functions used for regular expression matching: *re-compile* and *re-match*. *Re-compile* compiles a RE to a format that is more suitable for efficient matching. The compiled format is called a CRE (compiled RE):

```
(re-compile RE) => CRE
```

The *re-match* function matches a CRE against a sequence of characters. It returns the subsequence that matches the CRE or `:f` if the sequence does not match:

```
(re-match (re-compile '[a-z]*') '#__abc__') => '#abc'
(re-match (re-compile '[0-9]*') '#__abc__') => :f
```

The matcher uses a first-match and longest-match-first strategy. *First match* means that given multiple potential matches, it returns the first one:

```
(re-match (re-compile '[a-z]*') '#_abc_def_') => '#abc'
```

Longest match first means that each operator matching a subsequence (like `*` and `+`) attempts to match as many characters as possible (this method is also known as *eager* matching):¹⁷

```
(re-match (re-compile '#x*x') '#x__x__x') => '#x__x__x'
```

Another strategy (which is known as *shortest match first* or *lazy* matching) would attempt to find the shortest possible string matching an RE. Using this approach, the above expression would return `'#x__x'`.

As can be seen above, the `zenlisp` RE matcher uses lists of single-character symbols to represent sequences of characters. Of course, this means that only a limited character set can be used by it, but the underlying principles are the same as, for instance, in the `grep` utility.

Lacking the concept of a “line” of text, the `^` and `$` operators denote the beginning and end of a sequence in this section:

```
(re-match (re-compile '#[a-z]*') '#12test34) => '#test
(re-match (re-compile '#^12[a-z]*') '#12test34) => '#12test
(re-match (re-compile '#[a-z]*34$') '#12test34) => '#test34
(re-match (re-compile '#^[a-z]*$') '#12test34) => :f
```

9.3.1 regular expression compilation

The following list defines the character set on which the RE functions will operate. Characters replaced with `__` cannot be represented using symbols. The set is basically a subset of ASCII excluding the control character block and the symbols that are reserved for the `zenlisp` language. Because the interpreter does not distinguish between upper and lower case characters, they are considered to be equal.

```
(define character-set
  '( ( __ ! " __ $ % & __ __ __ * + , - __ /
    0 1 2 3 4 5 6 7 8 9 : __ < = > ?
    @ a b c d e f g h i j k l m n o
    p q r s t u v w x y z [ \ ] ^ _
    ` a b c d e f g h i j k l m n o
    p q r s t u v w x y z __ | __ ~ __ ) )
```

Pair-p is just a short cut.

```
(define (pair-p x) (not (atom x)))
```

The *before-p* predicate checks whether the character *c0* appears before the character *c1* in *character-set*. It will be used to check “-” operators in character classes.

```
(define (before-p c0 c1)
  (letrec
```

¹⁷ The longest match first approach is nowadays also called “greedy” matching, but I consider this term to be unfortunate, because it propagates a destructive mindset that already has caused enough suffering on our planet.

zen style programming

```
((lt (lambda (set)
      (cond ((null set) (bottom (list before-b c0 c1)))
            ((eq c1 (car set)) :f)
            ((eq c0 (car set)) :t)
            (t (lt (cdr set))))))
 (lt character-set)))
```

Make-range adds a new range (from *c0* through *cn*) to the class *cls*, e.g.:

```
(make-range 'a 'f '#9876543210) => '#fedcba9876543210
```

It is used to expand “-” operators in classes.

```
(define (make-range c0 cn cls)
  (letrec
    (make
      (lambda (c cls)
        (cond ((null c)
              (bottom 'invalid-symbol-code cn))
              ((eq (car c) cn)
               (cons (car c) cls))
              (t (make (cdr c)
                       (cons (car c) cls))))))
    (let ((c (memq c0 character-set)))
      (cond (c (make c cls))
            (t (bottom 'invalid-symbol-code c0))))))
```

The *compile-class* function compiles the character class at the beginning of the argument *in* and conses it to *out*. The *cls* argument holds the operator [that will be used to indicate a class in the resulting CRE. This operator will be changed to] when compiling complement classes (starting with [^). *First* is a flag that is initially set to “true” to indicate that *compile-class* currently processes the first character of the class. It is used to recognize ^ operators. The function returns a list containing the rest of its input as its first member and the compiled class as its second member:

```
(compile-class '#0-9] () '#[ :t) => '(() ([0123456789])
(compile-class '#^0-9] () '#[ :t) => '(() ([0123456789])
(compile-class '#0-9]xyz () '#[ :t) => '([xyz ([0123456789])
```

When invalid input is passed to *compile-class*, it returns :f:

```
(compile-class '#0-9 () '#[ :t) => :f ; missing ]
```

Note that the class operator itself ([) has to be consumed by the *caller* of *compile-class*.

```
(define (compile-class in out cls first)
  (cond
    ((null in) :f)
    ((eq ']' (car in))
     (list (cdr in) (cons (reverse cls) out)))
    ((and first (eq '^ (car in)))
     (compile-class (cdr in) out '#] :f))
```

```
((and (not first)
      (not (null (cdr cls)))
      (eq '- (car in))
      (pair-p (cdr in))
      (not (eq ']' (cadr in))))
  (let ((c0 (car cls))
        (cn (cadr in)))
    (cond
     ((before-p c0 cn)
      (compile-class (cddr in)
                     out
                     (make-range c0 cn (cdr cls)) :f))
     (t (compile-class (cdr in)
                       out
                       (cons '- cls) :f)))))
  (t (compile-class (cdr in)
                    out
                    (cons (car in) cls) :f))))
```

The *re-compile* function compiles an RE to a compiled RE (CRE). REs map to CREs as follows:

re	cre
[class]	'#[class
[^class]	'#]class
pattern*	(* pattern)
pattern+	pattern (* pattern)
pattern?	(? pattern)
^	#^
\$	#\$
\c	c

So, for example:

```
(re-compile '#\[a-c][^d-f]*\) => '(* #[abc (* #[abc] (* #[def] *)
```

Note that *pattern+* compiles to *pattern (* pattern)*; there is no separate CRE operator implementing *+*. *Re-compile* returns **:f** when an invalid RE is passed to it.

```
(define (re-compile re)
  (letrec
    ((compile
      (lambda (in out)
        (cond
         ((not in) :f)
         ((null in) (reverse out))
         (t (cond
              ((eq (car in) '\)
               (cond ((pair-p (cdr in))
                     (compile (cddr in)
                              (cons (cadr in) out)))
                    (t :f)))
              (t :f))))))
```

zen style programming

```

((memq (car in) '#^$_)
 (compile (cdr in)
  (cons (list (car in)) out)))
(memq (car in) '#*?)
 (compile (cdr in)
  (cond ((null out)
        (cons (car in) out))
        (t (cons (list (car in) (car out))
                  (cdr out))))))
(eq (car in) '+)
 (compile (cdr in)
  (cond ((null out)
        (cons (car in) out))
        (t (cons (list '* (car out)) out))))))
(eq (car in) '[])
 (apply compile
  (compile-class (cdr in) out '#[ :t)))
(t (compile (cdr in)
  (cons (car in) out))))))
(compile re ()))

```

9.3.1 regular expression matching

The *match-char* function matches a character (represented by a single-character symbol) against another character or a *class* of characters. For instance:

```

(match-char 'x 'x) => :t
(match-char '#]abc 'x) => :t
(match-char '_' 'x) => :t
(match-char '#[123 'x) => :f

```

When the pattern *p* matches the character *c*, it returns truth, else falsity.

```

(define (match-char p c)
  (cond ((eq '_ p)
        :t)
        ((atom p)
         (eq p c))
        ((eq '[' (car p))
         (and (memq c (cdr p)) :t))
        ((eq ']' (car p))
         (not (memq c (cdr p))))
        (t :f)))

```

Make-choices generates alternatives for matching subsequences using the *** operator. For example, when matching the pattern `[a-f] *` against the sequence `abc123`, the following alternatives exist:

```

(make-choices '((* #[abcdef]) '#abc123 ())
=> '((#abc123 ())
    (#bc123 #a)

```

```
(#c123 #ba)
(#123 #cba))
```

These alternatives are used to *backtrack* when the longest match causes the rest of the RE to mismatch. For instance when matching the RE `a*ab` against the sequence `aaaab`, the `a*` pattern could match `aaaa`, but then the subsequent pattern `ab` would not match `b`, so the match would fail. By backtracking, the matcher would then try to associate `a*` with `aaa`. In this case `ab` is matched against `ab`, so the whole RE matches:

```
(re-match (re-compile '#a*ab) '#aaaab) => '#aaaab
```

Note: The choices created by *make-choices* do include an empty match. The *m* argument of *make-choices* must be `()` initially.

The value returned by *make-choices* contains the rest of the sequence to be matched in its *car* part and the matched part of the sequence its *cadr* part. The matched part is returned in reverse order, because it will be passed to *match-cre* later in the process (see below).

```
(define (make-choices p s m)
  (cond
    ((or (null s)
         (not (match-char (cadr p) (car s))))
      (list (list s m)))
    (t (cons (list s m)
              (make-choices p (cdr s) (cons (car s) m))))))
```

The *match-star* function tries the alternatives generated by *make-choices* and finds the longest match that does not make the rest of the RE fail. Note that it returns the sequence matched by the complete remaining part of the CRE passed to it and not just the longest match found for the `*` operator at its beginning:

```
(match-star '((* a) a b) '#aaaab ()) => '#aaaab
```

Match-star reverses the result of *make-choices* because it lists the shortest match first.

```
(define (match-star cre s m)
  (letrec
    ((try-choices
      (lambda (c*)
        (cond ((null c*) :f)
              (t (let ((r (match-cre (cdr cre) (caar c*) (cadr c*)))
                    (cond (r (append (reverse m) r))
                          (t (try-choices (cdr c*)))))
                  (try-choices (reverse (make-choices cre s ())))))))))
```

The *match-cre* function matches all characters and operators except for `^`. It matches a compiled RE against the sequence *s*. Matching starts at the beginning of the sequence, so *match-cre* does *not* find occurrences of the RE that occur later in *s*:

```
(match-cre '(* #[ab] c) '#lab2 ()) => :f
```

zen style programming

An already-matched part may be passed to *match-cre* using the *m* argument, but it must be in reverse order, because *match-cre* conses to it and reverses *m* when it is done.

```
(define (match-cre cre s m)
  (cond
    ((null cre)
     (reverse m))
    ((null s)
     (cond ((equal cre '($))
            (match-cre () () m))
           ((and (pair-p (car cre))
                  (eq '* (caar cre))
                  (null (cdr cre)))
            ())
           (t :f)))
    ((pair-p (car cre))
     (cond
      ((eq '* (caar cre))
       (match-star cre s m))
      ((eq '? (caar cre))
       (cond ((match-char (cadar cre) (car s))
              (match-cre (cdr cre) (cdr s) (cons (car s) m)))
             (t (match-cre (cdr cre) s m))))
      ((match-char (car cre) (car s))
       (match-cre (cdr cre) (cdr s) (cons (car s) m)))
      (t :f)))
    ((eq (car cre) (car s))
     (match-cre (cdr cre) (cdr s) (cons (car s) m)))
    (t :f)))
```

Try-matches attempts to match a given RE to each tail of a given sequence, so unlike *match-cre* it also matches occurrences of the RE that begin later in the sequence:

```
(try-matches '((* #[ab] c) '#1abc2) => '#abc
```

Try-matches does not accept empty matches while iterating over the sequence. Only when the sequence has been completely visited without finding a non-empty match, an empty match is tried as a last resort.

```
(define (try-matches cre s)
  (cond ((null s) (match-cre cre s ()))
        (t (let ((r (match-cre cre s ())))
              (cond ((or (not r) (null r))
                     (try-matches cre (cdr s)))
                    (t r))))))
```

The *re-match* function matches the compiled regular expression *cre* against the sequence *s*. When the CRE starts with a ^ operator, *match-cre* is used to match the RE, otherwise *try-matches* is used.

```
(define (re-match cre s)
  (cond ((and (pair-p cre) (equal '#^ (car cre)))
        (match-cre (cdr cre) s ()))
        (t (try-matches cre s))))
```

How are the following REs interpreted? How should they be interpreted? (Q24)

```
(re-compile '#[-x])
(re-compile '#[x-])
(re-compile '#[])
(re-compile '#[^])
```

Why are regular expressions compiled and matched separately? Would it not be easier to compile and match them in one step?

The *make-choices* function creates all potential matches for a repetitive pattern, but only one of its results is actually used. Can you modify the code in such a way that it creates the next choice only if the current one does not match?

Can you turn the RE matcher introduced in this section into a shortest-match-first implementation? (Q25)

9.4 meta-circular interpretation

A *meta-circular* interpreter for a language **L** is an interpreter that is itself written in **L** and makes use of functions and other facilities provided by the host interpreter instead of re-implementing them. Meta-circular interpreters are capable of interpreting themselves, so whenever a meta-circular interpreter is run, there is an “outer” and an “inner” interpreter, i.e. an interpreter running the (inner) interpreter and an interpreter running a program. This principle is illustrated in figure 7.

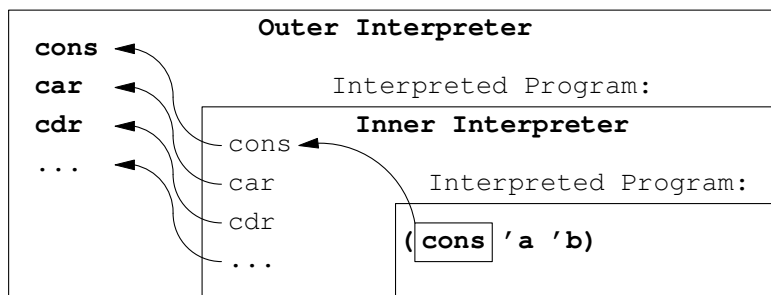


Fig. 7 – meta-circular interpretation

The inner interpreter is a meta-circular interpreter **M** while the outer one may be either another instance of **M** or a “native” interpreter.

This principle illustrates nicely that the question of “interpretation” versus “compilation” is an illusory one: a meta-circular interpreter is interpreted by a native interpreter, which is either

zen style programming

interpreted itself or “compiled”. When it is compiled, it is in fact interpreted by the CPU. The CPU instructions are interpreted by microcode, microcode is interpreted by transistors. Transistor functions are interpreted by the Laws of Nature. So unless your program is executed directly by the Laws of Nature, it is not really efficient, but probably more comprehensible.

A meta-circular interpreter for `zenlisp` would not implement functions like `cons` on its own but simply delegate applications of `cons` to the primitive `cons` function of the outer interpreter (see figure 7). Such an interpreter would not employ a parser, either, because `zenlisp` programs are `zenlisp` data, so the parser (reader) of the outer interpreter would be used instead.

The *zeval* function discussed in this section implements a meta-circular interpreter for `zenlisp` (modulo `define` and the meta functions). It accepts a program and an environment as its arguments and returns the normal form of the program in the given environment:

```
(zeval '(letrec
  (append
    (lambda (a b)
      (cond ((null a) b)
            (t (cons (car a)
                     (append (cdr a) b))))))
  (append '#hello- '#world!))
  (list (cons 'null null)))
=> '#hello-world!
```

Note that the `null` function has to be passed to *zeval* in the environment, because *zeval* implements only the core part of the language. Adding more functions is left as an exercise to the reader.

The code of *zeval* is not as simple as some other meta-circular interpreters that you may find in the literature, because it interprets tail recursive programs in constant space. Here is the code:

```
(define (zeval x e)
  (letrec
```

The *initial environment* of the interpreter contains the `zenlisp` keywords, the symbols `:t`, `:f`, and `t`, and a set of functions that cannot be implemented easily without referring to the outer interpreter. The initial environment may be considered a reasonable minimal set of symbols, keywords, and functions that is necessary to build `zenlisp`.

```
((initial-env
  (list (cons 'closure      'closure)
        (cons 't           ':t)
        (cons 't           ':t)
        (cons 'f           ':f)
        (cons 'and         '(%special . and))
        (cons 'apply       '(%special . apply))
        (cons 'cond        '(%special . cond))
        (cons 'eval        '(%special . eval))
        (cons 'lambda      '(%special . lambda))
        (cons 'let         '(%special . let))
```

```
(cons 'letrec      '(%special . letrec))
(cons 'or          '(%special . or))
(cons 'quote       '(%special . quote))
(cons 'atom        (cons '%primitive atom))
(cons 'bottom      (cons '%primitive bottom))
(cons 'car         (cons '%primitive car))
(cons 'cdr         (cons '%primitive cdr))
(cons 'cons        (cons '%primitive cons))
(cons 'defined     (cons '%primitive defined))
(cons 'eq          (cons '%primitive eq))
(cons 'explode     (cons '%primitive explode))
(cons 'implode     (cons '%primitive implode))
(cons 'recursive-bind (cons '%primitive recursive-bind)))
```

Value-of finds the value associated with a symbol in an association list and returns it. Unlike **assq**, it yields bottom when the symbol is not contained in the alist or associated with the special value %void. This function is used to look up values of variables in environments (which are implemented as alists).

```
(value-of
  (lambda (x e)
    (let ((v (assq x e)))
      (cond ((or (not v) (eq (cdr v) '%void))
             (bottom 'undefined: x))
            (t (cdr v))))))
```

Ev-list evaluates all members of *x* in the environment *e* and returns a list containing their normal forms. It is used to evaluate function arguments.

```
(ev-list
  (lambda (x e)
    (cond ((null x) ())
          ((atom x) (bottom 'improper-list-in-application: x))
          (t (cons (ev (car x) e)
                    (ev-list (cdr x) e))))))
```

Check-args checks whether the argument list *a* has *n* arguments (or $\geq n$, if the *more* flag is set). *Wrong-args* reports a wrong argument count. *Args-ok* is a front end to these functions. It is called by primitives to make sure that the correct number of arguments has been supplied.

```
(check-args
  (lambda (a n more)
    (cond ((null n) (or more (null a)))
          ((null a) :f)
          (t (check-args (cdr a)
                          (cdr n)
                          more)))))

(wrong-args
  (lambda (name args)
    (bottom 'wrong-number-of-arguments:
            (cons name args))))
```

zen style programming

```
(args-ok
  (lambda (name a n more)
    (cond ((check-args a n more) :t)
          (t (wrong-args name a)))))
```

The *eval-until* function evaluates the members of the list *a* in the environment *e*. As soon as a member of *a* evaluates to *t/f*, *eval-until* returns (**quote** *t/f*) immediately. When no member reduces to *t/f*, it returns the last member of *a* (*not* the normal form of that member).

Eval-until is used to implement the **and** and **or** keywords. It returns the last form in unevaluated state because evaluating it in situ would break tail recursion. This will become clear later in the code.

Note that *t/f* must be either **:t** or **:f**. When it is **:t**, it matches *any* “true” value due to the way it is checked in *eval-until*.

```
(eval-until
  (lambda (t/f a e)
    (cond ((null (cdr a)) (car a))
          ((atom a) (bottom 'improper-list-in-and/or: a))
          (t (let ((v (ev (car a) e)))
                (cond ((eq (not v) (not t/f))
                      (list 'quote v))
                  (t (eval-until t/f (cdr a) e))))))))
```

Do-and uses *eval-until* to implement **and**.

```
(do-and
  (lambda (a e)
    (cond ((null a) :t)
          (t (eval-until :f a e)))))
```

The *clause-p* and *do-cond* functions implement **cond**. *Clause-p* checks whether a clause of **cond** is syntactically correct.

Do-cond does a lot of checking first. The semantics of **cond** is implemented in the last clause, which loops through the clauses and returns the expression associated with the first “true” predicate. Like in *do-and* the expression is returned in unevaluated form.

```
(clause-p
  (lambda (x)
    (and (not (atom x))
         (not (atom (cdr x)))
         (null (cddr x)))))
(do-cond
  (lambda (a e)
    (cond ((null a)
          (bottom 'no-default-in-cond))
          ((atom a)
           (bottom 'improper-list-in-cond))
```

```
((not (clause-p (car a)))
 (bottom 'bad-clause-in-cond: (car a)))
(t (let ((v (ev (caar a) e)))
 (cond (v (cadar a))
 (t (do-cond (cdr a) e))))))
```

Do-eval implements **eval**. It merely passes its argument to the interpreter.

```
(do-eval
 (lambda (args e)
 (and (args-ok 'eval args '#i :f)
 (ev (car args) e))))
```

The *lambda-args* function turns an argument list (which may be a list, a dotted list, or even a symbol) into a proper list:

```
(lambda-args '(x y))    => '(x y)
(lambda-args '(x . y)) => '(x y)
(lambda-args 'x)        => '(x)
```

Its code is simple:

```
(lambda-args
 (lambda (a)
 (cond ((null a) ())
 ((atom a) (list a))
 (t (cons (car a)
 (lambda-args (cdr a)))))))
```

Add-free-var adds a binding for the free variable *var* to the environment *fenv*. The variable *e* is an environment in which *var* may be bound (it may be unbound as well). Here are some examples:

```
(add-free-var '((a . b)) 'a ()) => '((a . b))
(add-free-var () 'a '((a . x))) => '((a . x))
(add-free-var () 'a ())          => '((a . %void))
```

When the variable already is in *fenv* it is not added again. If it is not in *fenv* but in *e*, the binding of *e* is copied to *fenv*. If *var* is in neither environment, a new binding is created which associates the variable with the special symbol `%void`. This symbol indicates that the variable is not bound to any value.

```
(add-free-var
 (lambda (fenv var e)
 (cond ((assq var fenv) fenv)
 (t (let ((v (assq var e)))
 (cond (v (cons v fenv))
 (t (cons (cons var '%void) fenv)))))))
```

The *capture* function creates a snapshot of all free variables of the current environment. It is used to capture lexical environments when creating closures. *Bound* is a list of bound variables, *x* is the expression whose variables are to be captured, *e* is the currently active environment.

zen style programming

```
(capture
  (lambda (bound x e)
    (letrec
      ((collect
        (lambda (x free)
          (cond ((null x) free)
                ((atom x)
                 (cond ((memq x bound) free)
                       (t (add-free-var free x e))))
          (t (collect (car x)
                     (collect (cdr x) free)))))))
      (collect x ())))))
```

Do-lambda implements **lambda**. It returns a closure.

```
(do-lambda
  (lambda (args e)
    (and (args-ok 'lambda args '#ii :f)
         (list 'closure
               (car args)
               (cadr args)
               (capture (lambda-args (car args))
                        (cadr args)
                        e))))))
```

Do-or implements **or**. No surprise here.

```
(do-or
  (lambda (a e)
    (cond ((null a) :f)
          (t (eval-until :t a e)))))
```

Do-quote implements the **quote** keyword. It simply returns its argument. Note that the interpreter does not explicitly support '**x** in the place of (**quote x**). This is not necessary, because '**x** is expanded by the reader, so *zeval* never sees the unexpanded form.

```
(do-quote
  (lambda (args)
    (and (args-ok 'quote args '#i :f)
         (car args))))
```

The *make-env* function creates a new environment from the list of variables (formal arguments) *fa* and the (actual) argument list *aa*. It reduces to bottom when the numbers of arguments do not match. The function handles variadic argument lists correctly:

```
(make-env '(a . b) '(x y z)) => '((a . x) (b . (y z)))
(make-env 'a '(x y z))      => '((a . (x y z)))
```

Make-env is used to bind variables to arguments in function applications.

```
(make-env
  (lambda (fa aa)
```

```
(cond ((null fa)
      (cond ((null aa) ())
            (t (bottom 'too-many-arguments))))
      ((atom fa)
       (list (cons fa aa)))
      ((null aa)
       (bottom 'too-few-arguments))
      (t (cons (cons (car fa) (car aa))
                (make-env (cdr fa) (cdr aa))))))
```

The *beta* function implements *beta reduction* (see also page 59). It extends the current *outer* environment *e* by the union of the following partial environments:

- the bindings of the variables in *fa* to the arguments in *aa*;
- the lexical environment *lex-env*;
- the current *inner* environment *le* (local environment).

It also applies the *fix* function to the new local bindings formed by *fa* and *aa*. When *fix*=**id**, then *beta* implements ordinary beta reduction as in **lambda** and **let**. When *fix*=**recursive-bind**, it implements **letrec**.

Finally *beta* evaluates the expression *expr*. It passes both the inner (*e*) and outer environment (*le*) to *ev2*. This is done in order to implement tail recursion.

```
(beta
 (lambda (expr fa aa lex-env e le fix)
  (ev2 expr e (append (fix (make-env fa aa)) lex-env le))))
```

Do-let/rec implements **let** and **letrec**. It merely extracts the variables and arguments from the constructs and passes them to *beta*. *Le* and *e* are the current inner and outer environments. The *fix* parameter is explained above.

The *binding-p* helper makes use of the fact that clauses have the same syntax as bindings.

```
(binding-p
 (lambda (x)
  (clause-p x)))
(do-let/rec
 (lambda (args e le fix)
  (cond ((not (args-ok 'let/letrec args '#ii :f)) :f)
        ((not (apply and (map binding-p (car args))))
         (bottom 'bad-let/letrec-syntax: (car args)))
        (t (let ((formals (map car (car args)))
                  (actuals (map cadr (car args))))
              (beta (cadr args)
                    formals
                    (ev-list actuals le)
                    ()
                    e le fix))))))
```

zen style programming

Apply-fn applies the function *fn* to the formal arguments *args*. The arguments already are in their normal forms at this point. The environments *e* and *le* are merely passed through.

```
(apply-fn
  (lambda (fn args e le)
    (cond ((eq (car fn) '%primitive)
           (apply (cdr fn) args))
          ((eq (car fn) '%special)
           (apply-special (cdr fn) args e le))
          ((eq (car fn) '%closure)
           (beta (caddr fn)
                  (cadr fn)
                  args
                  (caddrn fn)
                  e le id))
          (t (bottom 'application-of-non-function: fn))))))
```

The *make-args* function creates an argument list for **apply**. Because **apply** itself is variadic, it collects the optional arguments and makes sure that the last one is a list:

```
(make-args '(a b c '(d e))) => '#abcde
(make-args '(a b c d))      => bottom
```

Here comes the code:

```
(make-args
  (lambda (a)
    (cond ((null (cdr a))
           (cond ((atom (car a))
                  (bottom 'improper-argument-list:
                          (car a)))
                 (t (car a))))
          (t (cons (car a) (make-args (cdr a)))))))
```

Apply-special interprets the keywords of **zenlisp** (except for **define**). It applies the pseudo function *fn* to the arguments *args*. Because *fn* is a pseudo function, the arguments are unevaluated and *not* in their normal forms.

Note that some of the cases of *apply-special* pass the value returned by a special form handler to *ev2* and some simply return it. This is because some special form handlers (for **and**, **cond**, etc) return expressions that still have to be reduced to their normal forms while others return normal forms immediately (like the handlers for **lambda**, **quote**, etc).

Still others (like the handlers for **let**, **letrec** and **apply**) call *ev2* themselves, so they recurse indirectly. Because handlers like *do-let/rec* are called in tail positions, recursion still happens in constant space.

```
(apply-special
  (lambda (fn args e le)
```

```
(cond ((eq fn 'and)
      (ev2 (do-and args le) e le))
      ((eq fn 'apply)
      (let ((args (ev-list args le)))
        (and (args-ok 'apply args '#ii :t)
              (apply-fn (car args)
                        (make-args (cdr args))
                        e
                        e))))
      ((eq fn 'cond)
      (ev2 (do-cond args le) e le))
      ((eq fn 'eval)
      (ev2 (do-eval args le) e le))
      ((eq fn 'lambda)
      (do-lambda args le))
      ((eq fn 'let)
      (do-let/rec args e le id))
      ((eq fn 'letrec)
      (do-let/rec args e le recursive-bind))
      ((eq fn 'or)
      (ev2 (do-or args le) e le))
      ((eq fn 'quote)
      (do-quote args))
      (t (bottom 'internal:bad-special-operator: fn))))
```

These predicates check whether an object is a function or a special form handler.

```
(function-p
  (lambda (x)
    (or (eq (car x) '%primitive)
        (eq (car x) 'closure))))
(special-p
  (lambda (x)
    (eq (car x) '%special)))
```

Ev2 reduces the expression *x* to its normal form in the environment *le*. The argument *e* holds the current *outer* environment. Initially, *e* equals *le*.

In order to find out what to do with a list, *ev2* reduces the car part of the list in *le*. If the resulting normal form *f* is a function, it evaluates the function arguments (the cdr part of the list) in *le* and then discards the inner environment by setting *new-e=e*. When *f* is a special form handler, function arguments are not evaluated and the inner environment is kept.

Abandoning the inner environment after evaluating function arguments implements tail recursion.

```
(ev2
  (lambda (x e le)
    (cond
      ((null x) ())
      ((atom x) (value-of x le))
```


zen style programming

```
(t (let ((f (ev (car x) le)))
      (cond ((eq f 'closure) x)
            ((atom f)
             (bottom 'application-of-non-function: f))
            (t (let ((args (cond ((function-p f)
                                   (ev-list (cdr x) le))
                                (t (cdr x)))))
                  (new-e (cond ((special-p f) le)
                                (t e))))
                  (apply-fn f args e new-e)))))))
```

This is just an abbreviation for the case $e=le$.

```
(ev (lambda (x e)
      (ev2 x e e)))
```

The environment passed to *zeval* is attached to the initial environment before starting the interpreter:

```
(ev x (append e initial-env)))
```

Can you rewrite *zeval* in such a way that it actually can interpret itself?

Primitives are just passed through to the host interpreter, e.g. **cons** is interpreted by the code implementing **cons** in the outer interpreter. Why is it not possible to delegate evaluation of special forms to the special form handlers of the outer interpreter? (**Q26**)

Zeval uses symbols like %special and %void to tag special values. Could this detail cause any trouble when programs to be interpreted by *zeval* contained such symbols? What can you do about it? (**Q27**)

10. mexprc – an m-expression compiler

This chapter will use some of the techniques introduced in the previous chapter to implement a compiler for a full programming language.

M-expressions (meta expressions) form the syntax that was originally intended for LISP. In the original design, *S-expressions* were only used for the representation of data, while M-expressions were used to write programs. An *S-expression* is basically equal to a `zenlisp` form.

Here is a quote from the ACM paper “History of LISP”¹⁸ by John McCarthy that explains why S-expressions were eventually used in the place of M-expressions:

S.R. Russel noticed that eval could serve as an interpreter for LISP, promptly hand coded it, and we now had a programming language with an interpreter. The unexpected appearance of an interpreter tended to freeze the form of the language [...]
The project of defining M-expressions precisely and compiling them or at least translating them to S-expressions was neither finalized nor explicitly abandoned. It just receded into the indefinite future, and a new generation of programmers appeared who preferred internal notation to any FORTRAN-like or ALGOL-like notation that could be devised.

So the syntax of M-expressions was never specified precisely. However, many LISP texts make use of M-expressions, so enough information can be gathered from them to construct a language that probably comes close to what M-expressions would have been. This chapter defines a BNF grammar for M-expressions and implements a compiler that translates M-expressions to S-expressions (`zenlisp` programs).

10.1 specification

M-expressions are similar to the infix notation that is employed by FORTRAN, C, Java, and other programming languages. Numbers represent themselves and the usual operators are used to express math operations such as subtraction, multiplication, etc. Due to the lack of suitable characters in the ASCII character set, logical AND and OR is represented by the sequences `/\` and `\/` respectively and the right arrow is written as `->`.

“Real” M-expressions would make use of the characters `(,)`, and `;`, but the MEXPRC compiler cannot implement them in this way, because these characters cannot be contained as data in `zenlisp` programs. Hence the specification has to be adjusted a bit:

¹⁸ The full paper can be found at McCarthy’s homepage:
<http://www-formal.stanford.edu/jmc/history/lisp/lisp.html>

zen style programming

- expression grouping is done by [and] rather than (and);
- literal lists are delimited by << and >> rather than (and);
- function arguments are separated using , instead of ;;
- the conditional operator is written [a->b:c] rather than [a->b;c];
- constants are prefixed with % instead of using upper case.

Here are some sample M-expressions and their corresponding S-expressions:

cons[a,b]	(cons a b)
a::b	(cons a b)
%a::%b	(cons 'a 'b)
%a::<<b,c,d>>	(cons 'a '(b c d))
append[a,b]	(append a b)
a++b	(append a b)
a*b-c/d	(- (* a b) (/ c d))
[a+b]*c	(* (+ a b) c)
[a/\b-> c: d]	(cond ((and a b) c) (t d))
f[x] := x^2	(define (f x) (expt x 2))
lambda[[x] cons[x,x]]	(lambda (x) (cons x x))
lambda[[x] x][%a]	((lambda (x) x) 'a)
not[x] :=	(define (not x)
[x-> false: true]	(cond (x :f) (t :t)))
fact[x] :=	(define (fact x)
[x=0	(cond ((= x 0)
-> 1:	1)
fact[x-1]*x]	(t (* (fact (- x 1)) x)))
f[x] := g[x]	(define (f x)
where g[x] := h[x]	(letrec ((g (lambda (x) (h x)))
and h[x] := x	(h (lambda (x) x)))
	(g x)))

The complete syntax of the source language accepted by MEXPRC is described in the BNF grammar following below. The grammar uses the *concatenation operator* & which is not typically found in BNF grammars. Rules of the form

a* := a | a & a*

match only sequences of as that do not contain any blank characters in between, e.g. the above

rule matches

```
aaaaaa
```

but not

```
a a a a a a
```

Concatenation operators (&) bind stronger than OR operators (|). They are used to introduce token classes like sequences of digits representing numbers or sequences of characters representing symbols.

Some rules of the grammar are annotated with S-expressions that indicate what the sentence matched by the rule is to be translated to. Such annotations are common for specifying the semantics of a language semi-formally. For instance the rule

```
concatenation :=  
  factor '::' concatenation ; (cons factor concatenation)
```

states that each sentence of the above form denotes an application of **cons**. Like `zenlisp` comments, annotations are introduced by a semicolon and extend up to the end of the current line.

10.1.1 annotated grammar

Note: non-terminals have no angle brackets in this grammar. All terminals are enclosed in apostrophes.

```
mexpr := definition  
      | expression  
  
numeric-char := '0' | ... | '9'  
  
symbolic-char := 'a' | ... | 'z' | '_'  
  
number := numeric-char  
        | number & numeric-char  
  
symbol := symbolic-char  
        | symbol & symbolic-char  
  
list-member := symbol  
            | list  
  
list-members := list-member  
              | list-member list-members  
  
list := '<<' list-members '>>' ; (quote (list-members))  
      | '<<' '>>' ; ()  
  
list-of-expressions := expression  
                    | expression ',' list-of-expressions
```

zen style programming

```
list-of-symbols := symbol
                  | symbol ',' list-of-symbols

cases := case
        | case ':' cases

case := expression '->' expression

factor :=
    number                ; number
  | symbol                 ; variable
  | '%' symbol             ; (quote symbol)
  | 'true'                 ; :t
  | 'false'                ; :f
  | '-' factor             ; (- factor)
  | symbol '[' list-of-expressions ']' ; (symbol list-of-expressions)
  | symbol '[' ']'         ; (symbol)
  | '[' expression ']'     ; expression
  | '[' cases ':' expression ']' ; (cond cases (t expression))
                                ; where cases
                                ; = ((expression expression) ...)
                                ; as in "case"
  | lambda                 ; (lambda ...)
  | lambda '[' list-of-expressions ']' ; ((lambda ...) list-of-expressions)
  | lambda '[' ']'         ; ((lambda ...))

lambda :=
  'lambda' '[' '[' list-of-symbols ']' expression ']'
    ; (lambda (list-of-symbols) expression)
  | 'lambda' '[' '[' ']' expression ']'
    ; (lambda () expression)

concatenation :=
  factor
  | factor ':' concatenation ; (cons factor concatenation)
  | factor '++' concatenation ; (append factor concatenation)

power := concatenation
        | concatenation '^' power ; (expt concatenation power)

term := power
        | term '*' power ; (* term power)
        | term '/' power ; (/ term power)
        | term '//' power ; (quotient term power)
        | term '\\\' power ; (remainder term power)

sum := term
      | sum '+' term ; (+ sum term)
      | sum '-' term ; (- sum term)
```

```

predicate := sum '=' sum      ; (= sum sum)
           | sum '<>' sum      ; (not (= sum sum))
           | sum '<' sum       ; (< sum sum)
           | sum '>' sum       ; (> sum sum)
           | sum '<=' sum      ; (<= sum sum)
           | sum '>=' sum      ; (>= sum sum)
           | sum

conjunction :=
  predicate
  | conjunction '/'\ predicate ; (and conjunction predicate)

disjunction :=
  conjunction
  | disjunction '/'\ conjunction ; (or disjunction conjunction)

expression := disjunction

definition :=
  simple-definition
  | simple-definition 'where' definition-list
    ; (define ... (letrec definition-list ...))
    ; where definition-list
    ;           = ((symbol (lambda (list-of-symbols) expression)) ...)
    ;           | ((symbol (lambda () expression)) ...)
    ; as in "simple-definition"

simple-definition :=
  symbol '[' list-of-symbols ']' := expression
    ; (define (symbol list-of-symbols) expression)
  | symbol '[' ']' := expression
    ; (define (symbol) expression)

definition-list := simple-definition
                 | simple-definition 'and' definition-list

```

10.2 implementation

We will dive right into the code.

The rational math package is required because of the / operator.

```
(require '~rmath)
```

These classes contain the characters used to compose symbols and numbers:

```
(define symbol-class '#abcdefghijklmnopqrstuvwxyz_)
```

```
(define number-class '#0123456789)
```

Symbol-p and *number-p* check whether a character (represented by a single-character symbol)

zen style programming

belongs to a specific class:

```
(define (symbol-p x) (and (memq x symbol-class) :t))

(define (number-p x) (and (memq x number-class) :t))
```

10.2.1 lexical analysis

The *prefix->infix* parser [page 117] used lists of symbols to represent input programs. Because M-expressions may be longer than a few characters and may span multiple lines of input, this representation is not suitable for M-expressions, though.

Instead of lists of single-character symbols, MEXPRC will use lists of symbols of variable length. Each symbol may contain one or multiple tokens. Here is a sample M-expression in MEXPR notation:

```
'( fact[x] :=
    [x=0-> 1:
      fact[x-1]*x] )
```

This list contains the following symbols

```
fact [x]
:=
[x=0->
1:
fact[x-1]*x]
```

Each of the symbols contains at least one *token*. *Token* is just another word for *lexeme*. It is quite common in compiler texts. These are the individual tokens contained in the above list:

```
fact [ x ]
:=
[ x = 0 ->
1 :
fact [ x - 1 ] * x ]
```

A symbol containing some tokens is called a *fragment*, because it holds a fragment of an input program. Each M-expression program is represented by a sequence of fragments.

The first stage of the MEXPRC compiler scans such a sequence of fragments and decomposes them into individual tokens. This process is called *lexical analysis*. The output of this stage is a sequence of tokens:

```
(tokenize '(fact[x] := [x=0-> 1: fact[x-1]*x]))
=> '(fact [ x ] := [ x = 0 -> 1 : fact [ x - 1 ] * x ])
```

Fragments are exploded when they are encountered in the input stream for the first time:

```
(define (explode-on-demand fragment)
  (cond ((atom fragment) (explode fragment))
        (t fragment)))
```

Extract-class extracts a sequence matching a character class from the front of a fragment.

```
(define (extract-class fragment class-p)
  (letrec
    ((input
      (explode-on-demand fragment))
     (x-class
      (lambda (input sym)
        (cond ((null input)
              (list (reverse sym) input))
              ((class-p (car input))
               (x-class (cdr input)
                        (cons (car input) sym)))
              (t (list (reverse sym) input))))))
    (x-class input ())))
```

The following functions extract symbols and numbers from the current fragment. Like most lexical analysis functions, they return a list containing the extracted token and the remaining input, both in exploded form:

```
(extract-symbol 'abc+def) => '(#abc #+def)
```

The functions are implemented on top of *extract-class*:

```
(define (extract-symbol fragment)
  (extract-class fragment symbol-p))

(define (extract-number fragment)
  (extract-class fragment number-p))
```

Extract-char simply extracts the leading character from the input:

```
(define (extract-char fragment)
  (let ((input (explode-on-demand fragment)))
    (list (list (car input)) (cdr input))))
```

The *extract-alternative* function extracts a single- or double-character token from the head of a fragment. If the second character of the fragment is contained in the *alt-tails* argument, a two-character token is extracted and else a single character is extracted. For instance:

```
(extract-alternative '>=xyz '#>=) => '(#>= #xyz)
(extract-alternative '>-xyz '#>=) => '(#> #-xyz)
```

The function is used to extract operator symbols.

```
(define (extract-alternative fragment alt-tails)
  (let ((input (explode-on-demand fragment)))
```


zen style programming

```
(cond ((null (cdr input))
      (extract-char input))
      ((memq (cadr input) alt-tails)
       (list (list (car input) (cadr input))
             (caddr input)))
      (t (extract-char input))))
```

Extract-token extracts any kind of token by dispatching its input to one of the above functions. When the first character of the current fragment cannot be identified, the function signals a syntax error.

```
(define (extract-token fragment)
  (let ((input (explode-on-demand fragment)))
    (let ((first (car input)))
      (cond ((eq first '[')
             (extract-char input))
            ((eq first ']')
             (extract-char input))
            ((eq first ',')
             (extract-char input))
            ((eq first '%')
             (extract-char input))
            ((eq first ':')
             (extract-alternative input '#:=))
            ((eq first '+')
             (extract-alternative input '#+))
            ((eq first '-')
             (extract-alternative input '#>))
            ((eq first '*')
             (extract-char input))
            ((eq first '=)
             (extract-char input))
            ((eq first '<')
             (extract-alternative input '#<=>))
            ((eq first '>')
             (extract-alternative input '#>=))
            ((eq first '/')
             (extract-alternative input '#/\'))
            ((eq first '\')
             (extract-alternative input '#/\'))
            ((eq first '^')
             (extract-char input))
            ((symbol-p first)
             (extract-symbol input))
            ((number-p first)
             (extract-number input))
            (t (bottom 'syntax 'error 'at input))))))
```

These are just some more comprehensible names for members of the intermediate format used during lexical analysis:

```
(define frag car)      ; fragment of input
(define rest cdr)      ; rest of input
(define restfrag cadr) ; fragment of rest of input
(define restrest cddr) ; rest of rest of input
```

The *next-token* function extracts the first token of the first fragment of a fragment list. If the first fragment is empty, it removes it and advances to the next fragment.

```
(define (next-token source)
  (cond ((null (frag source))
        (cond ((null (rest source)) ())
              (t (let ((head (extract-token (restfrag source)))
                      (cons (implode (frag head))
                            (cons (restfrag head)
                                  (restrest source)))))))
        (t (let ((head (extract-token (frag source)))
                (cons (implode (frag head))
                      (cons (restfrag head)
                            (rest source)))))))))
```

The *tokenize* function forms the front end of the *scanner* of the MEXPR compiler. The *scanner* is the part of a compiler that performs lexical analysis. *Tokenize* accepts a source program and emits a list of separate tokens.

```
(define (tokenize source)
  (letrec
    ((tok (lambda (src tlist)
            (let ((new-state (next-token src)))
              (cond ((null new-state) (reverse tlist))
                    (t (tok (cdr new-state)
                           (cons (car new-state)
                                 tlist)))))))
    (tok source ())))
```

10.2.2 syntax analysis and code synthesis

Syntax analysis and *code synthesis* are interleaved in the following code. Zenlisp expressions are synthesized *while* performing syntax analysis.

The parser is the controlling instance of the MEXPRC compiler: it reads input through the scanner and emits code through the synthesizer (which in this case is not even a separate stage). This is why this approach is called *syntax-directed compilation*.

The parsing stage of the compiler accepts input in the form of a token list (as generated by the scanner) and emits a zenlisp expression:

```
(mexpr-compile (f [ x ] := 2 ^ x) => (define (f x) (expt '#2 x))
```

Most functions of the syntax analysis stage of the compiler process partially translated programs of the form

zen style programming

(S-expression token-list)

where *S-expression* is a *zenlisp* program under construction and *token-list* is the not yet translated rest of the input program. Initially the *S-expression* part is empty and the *token-list* part contains the full tokenized input program. During compilation, data is moved from the *token-list* to the *S-expression* part and when compilation finishes successfully, the *token-list* part is empty and the *S-expression* part contains a complete *zenlisp* expression.

The *parse-term* function, for instance, parses and synthesizes a term:

```
(parse-term '(() #a*b+c)) => '((* a b) '#+c)
```

Initially, the synthesized expression is *()* and the input program is *'#a*b+c*. *Parse-term* removes the term *'#a*b* and generates a program structure containing the synthesized expression *'(* a b)* and the “rest” *'#+c*.

Make-prog composes a program structure:

```
(define (make-prog sexpr tlist) (list sexpr tlist))
```

These functions are used to access the individual parts of a program structure:

```
(define s-expr-of car)      ; S-expression built so far
```

```
(define rest-of cadr)      ; Not yet translated rest of program
```

Has the end of the input program been reached?

```
(define (end-of p) (null (rest-of p)))
```

This function delivers the current input token or *()*, if input is exhausted.

```
(define (first-of-rest p)
  (cond ((end-of p) ())
        (t (caadr p))))
```

Rest-of-rest delivers the input program with its first token removed. It is used to advance to the next input token.

```
(define (rest-of-rest p)
  (cond ((end-of p) ())
        (t (cdadr p))))
```

Look-ahead and *rest-of-look-ahead* are similar to *first-of-rest* and *rest-of-rest*, but produce the second token in the input and the rest following the second token respectively:

```
(first-of-rest      '(() #a+b*c)) => 'a
(rest-of-rest      '(() #a+b*c)) => '#+b*c
(look-ahead        '(() #a+b*c)) => '+'
(rest-of-look-ahead '(() #a+b*c)) => '#b*c
```

The second character in the input is called the *look-ahead* token. It is used to figure out what kind of input is following when the first token is ambiguous. For instance, a symbol name may or may not be followed by an opening parenthesis. When there is an opening parentheses, the symbol is part of a function application (or definition) and otherwise it is a reference to a variable:

```
'(sym + sym)
'(sym [ arg ])
```

In this case, the look-ahead token can be used to delegate further analysis to the appropriate procedure.

```
(define (look-ahead p)
  (cond ((end-of p) ())
        ((null (rest-of-rest p)) ())
        (t (car (rest-of-rest p)))))

(define (rest-of-look-ahead p)
  (cond ((end-of p) ())
        ((null (rest-of-rest p)) ())
        (t (cdr (rest-of-rest p)))))
```

Extract first char of a token

```
(define (first-char x) (car (explode x)))
```

Quoted quotes a form.

```
(define (quoted x) (list 'quote x))
```

Parse-list is the first function that actually performs some syntax analysis. It accepts an M-expression representing a literal list and returns a quoted list in S-expression form, for example:

```
<< a, b, <<c>>, d>>  →  (quote (a b (c) d))
```

(The arrow denotes the transformation of an M-expression to an S-expression in this section.)

The embedded *plist* function, which performs the actual work, has four parameters with the following meanings: *tls* is the input token list, *lst* is the output list constructed so far, *skip* is used to skip over commas, and *top* is a flag indicating whether we are currently parsing a top level list (as opposed to a nested list).

The function recurses to parse embedded lists.

```
(define (parse-list tlist)
  (letrec
    ((plist
      (lambda (tls skip lst top)
        ; tls = input
        ; skip = skip next token (commas)
        ; lst = output
        ; top = processing top level list
```

zen style programming

```
(cond ((eq (car t1s) '>>)
      (cond (top (make-prog (quoted (reverse 1st))
                            (cdr t1s)))
            (t (make-prog (reverse 1st)
                          (cdr t1s)))))
      ((eq (car t1s) '<<)
       (let ((sublist (plist (cdr t1s) :f () :f)))
         (plist (rest-of sublist)
                 :t
                 (cons (car sublist) 1st)
                 top)))
      (skip
       (cond ((eq (car t1s) ',)
              (plist (cdr t1s) :f 1st top))
             (t (bottom ', 'expected 'at t1s))))
      (t (plist (cdr t1s)
                :t
                (cons (car t1s) 1st)
                top))))))
(plist tlist :f () :t)))
```

The following function reports an unexpected end of input. This is just an abbreviation that is being introduced because an unexpected EOT (end of text) is a common condition.

```
(define (unexpected-eot)
  (bottom 'unexpected-end-of-input))
```

The *parse-actual-args* function parses a list of function arguments (a list of expressions) and returns an equivalent list of S-expressions:

[a+b, c*d] → ((+ a b) (* c d))

It is used in applications of named functions and lambda functions.

```
(define (parse-actual-args tlist)
  (letrec
    ((pargs
      (lambda (t1s skip 1st)
        (cond ((null t1s) (unexpected-eot))
              ((eq (car t1s) ']')
               (make-prog (reverse 1st) (cdr t1s)))
              (skip
               (cond ((eq (car t1s) ',)
                      (pargs (cdr t1s) :f 1st))
                     (t (bottom ', 'expected 'at t1s))))
              (t (let ((expr (parse-expr t1s)))
                   (pargs (rest-of expr)
                           :t
                           (cons (car expr) 1st)))))))
    (pargs tlist :f ())))
```

Parse-formal-args parses the formal argument list of a function (the variables of a function):

$[a,b,c] \rightarrow (a\ b\ c)$

It is similar to *parse-actual-args* but accepts a list of symbols rather than a list of expressions.

```
(define (parse-formal-args tlist)
  (letrec
    ((pargs
      (lambda (tls skip lst)
        (cond ((null tls) (unexpected-eot))
              ((eq (car tls) ',)
               (make-prog (reverse lst) (cdr tls)))
              (skip
               (cond ((eq (car tls) ',)
                      (pargs (cdr tls) :f lst))
                     (t (bottom ', 'expected 'at tls))))
              ((symbol-p (first-char (car tls)))
               (pargs (cdr tls) :t (cons (car tls) lst)))
              (t (bottom 'symbol 'expected 'at tls))))))
    (pargs tlist :f ())))
```

Parse-fun-call parses a function application:

$f[a,b,c] \rightarrow (f\ a\ b\ c)$

It is used to parse applications of named functions exclusively.

```
(define (parse-fun-call program)
  (let ((function (first-of-rest program))
        (args (parse-actual-args (rest-of-look-ahead program))))
    (make-prog (append (list function)
                        (s-expr-of args))
                (rest-of args))))
```

Parse-lambda-args parses the formal argument list of a lambda function, if it follows in the input stream. If no argument list follows, report an error.

```
(define (parse-lambda-args program)
  (cond ((eq (first-of-rest program) '[])
         (parse-formal-args (rest-of-rest program)))
        (t (bottom 'argument 'list 'expected 'in 'lambda[]))))
```

Create a lambda function.

```
(define (make-lambda args term)
  (list 'lambda args term))
```

The *parse-lambda-app* function parses the application of a lambda function:

$[a_1 \dots a_n] \rightarrow ((\text{lambda } \dots) a_1 \dots a_n)$

When this function is entered, a leading lambda function has just been parsed and its code already

zen style programming

is in the S-expression part of the program structure passed to it:

```
(parse-lambda-app '((lambda (x) x) #[y]+z)) => '(((lambda (x) x) y) #+z)
```

Parse-lambda-app is invoked when a lambda function in the source program is followed by an opening parenthesis.

```
(define (parse-lambda-app program)
  (let ((args (parse-actual-args (rest-of-rest program))))
    (make-prog (append (list (s-expr-of program))
                        (s-expr-of args))
              (rest-of args)))))
```

Parse-lambda parses a lambda function:

$\text{lambda}[[v_1 \dots v_n] \text{ term}] \rightarrow (\text{lambda } (v_1 \dots v_n) \text{ term})$

It uses the look-ahead token for convenience. No actual look ahead is necessary at this point.

```
(define (parse-lambda program)
  (cond ((neq (look-ahead program) '[])
        (bottom '[ 'expected 'after 'lambda]))
        (t (let ((args (parse-lambda-args
                          (make-prog
                           ()
                           (rest-of-look-ahead program))))))
              (let ((term (parse-expr (rest-of args))))
                (cond ((neq (first-of-rest term) '[])
                      (bottom 'missing 'closing ']' in 'lambda[]))
                      (t (make-prog
                          (make-lambda (s-expr-of args)
                                         (s-expr-of term))
                          (rest-of-rest term))))))))))
```

Create a clause for **cond**:

```
(define (make-case pred expr) (list pred expr))
```

Parse-cases parses the cases of a conditional expression and generates a set of clauses for **cond**:

$a \rightarrow b: c \rightarrow d: e \rightarrow ((a \ b) \ (c \ d) \ (t \ e))$

Note that the function parses just the cases without the brackets that enclose a conditional M-expression.

```
(define (parse-cases program)
  (letrec
    ((pcases
      (lambda (prog cases)
        (let ((pred (parse-disj (make-prog () prog))))
          (cond
```

```
((neq (first-of-rest pred) '->)
  (make-prog (cons (make-case 't (s-expr-of pred))
                  cases)
             (rest-of pred)))
(t (let ((expr (parse-expr (rest-of-rest pred))))
  (cond
   ((eq (first-of-rest expr) ':)
    (pcases (rest-of-rest expr)
             (cons (make-case (s-expr-of pred)
                              (s-expr-of expr))
                   cases)))
   (t (bottom ': 'expected 'in 'conditional 'before
              (rest-of expr)))))))
(let ((case-list (pcases (rest-of program) ())))
  (make-prog (reverse (s-expr-of case-list))
             (rest-of case-list))))
```

Make-cond-expr creates a **cond** expression from a list of cases. When there is only one case (the default), skip **cond** and just synthesize the expression constituting that case.

This is a necessary extension and not just a performance hack, because *parse-cond-expr* (below) exploits the fact that a conditional expression with just a default case is syntactically equal to grouping: See *parse-cond-expr* for details.

```
(define (make-cond-expr cases)
  (cond ((null (cdr cases))
        (cadar cases))
        (t (cons 'cond cases))))
```

Parse-cond-expr parses a conditional expression and generates an equivalent **cond** expression:

$$[p_1 \rightarrow x_1: p_2 \rightarrow x_2: \dots x_n] \rightarrow (\text{cond } (p_1 \ x_1) \ (p_2 \ x_2) \ \dots \ (t \ x_n))$$

In fact, it merely skips over the opening square bracket, delegates analysis of the cases to *parse-cases* (above) and then makes sure that there is a delimiting closing bracket.

Parse-cond-expr handles both conditional expressions and grouping (because they are syntactically equal):

$$\begin{aligned} [\text{pred} \rightarrow \text{expr}: \text{expr}] &\rightarrow (\text{cond } (\text{pred } \text{expr}) \ (t \ \text{expr})) \\ [\text{expr}] &\rightarrow (\text{cond } (t \ \text{expr})) \rightarrow \text{expr} \end{aligned}$$

The redundant **cond** is removed by *make-cond-expr* (above).

```
(define (parse-cond-expr program)
  (let ((cond-expr
        (parse-cases
         (make-prog () (rest-of-rest program)))))
    (cond ((neq (first-of-rest cond-expr) ']')
           (bottom ']' 'expected 'at 'end 'of
```


zen style programming

```
          'conditional 'expression))
(t (make-prog
  (make-cond-expr (s-expr-of cond-expr))
  (rest-of-rest cond-expr))))))
```

Because *parse-cond-expr* has two functions, it should have two names that reflect these functions:

```
(define parse-grouped-expr parse-cond-expr)
```

The *parse-factor* function accepts a single factor of an M-expression and generates an equivalent S-expression. When the token at the beginning of the source part of the program structure *program* is not a valid factor, a syntax error is reported.

Transformations are inlined in the below code.

```
(define (parse-factor program)
  (let ((first (first-char (first-of-rest program))))
    (cond ((null first)
           (unexpected-eot))

          ; nil → ()
          ((eq (first-of-rest program) 'nil)
           (make-prog () (rest-of-rest program)))

          ; true → :t
          ((eq (first-of-rest program) 'true)
           (make-prog :t (rest-of-rest program)))

          ; false → :f
          ((eq (first-of-rest program) 'false)
           (make-prog :f (rest-of-rest program)))

          ; lambda[[v ...] x] → (lambda (v ...) x)
          ; lambda[[v ...] x][a] → ((lambda (v ...) x) a)
          ((eq (first-of-rest program) 'lambda)
           (let ((lambda-term (parse-lambda program)))
             (cond ((eq (first-of-rest lambda-term) '[])
                    (parse-lambda-app lambda-term))
                   (t lambda-term))))

          ; symbol → symbol
          ; symbol[x ...] → (symbol x ...)
          ((symbol-p first)
           (cond ((eq (look-ahead program) '[])
                  (parse-fun-call program))
                 (t (make-prog (first-of-rest program)
                                (rest-of-rest program))))))
```

```
; number → (quote #number)

((number-p first)
 (make-prog (quoted
             (explode
              (first-of-rest program)))
             (rest-of-rest program)))

; << element, ... >> → (quote (element ...))

((eq (first-of-rest program) '<<)
 (parse-list (rest-of-rest program)))

; %symbol → (quote symbol)

((eq first '%)
 (cond ((symbol-p (first-char (look-ahead program)))
        (let ((rhs (parse-factor
                     (make-prog
                      ()
                      (rest-of-rest program)))))
          (make-prog (quoted (s-expr-of rhs))
                     (rest-of rhs))))
        (t (bottom 'symbol 'expected 'after ':%: program))))

; [expression] → expression

((eq first '[])
 (parse-grouped-expr program))

; -factor → (- factor)

((eq first '-')
 (let ((rhs (parse-factor
               (make-prog
                ()
                (rest-of-rest program)))))
      (make-prog (list '- (s-expr-of rhs))
                 (rest-of rhs))))

(t (bottom 'syntax 'error 'at (rest-of program)))))
```

The *parse-binary* function parses all kinds of left-associative binary operators:

```
x op y op z → (funop (funop x y) z)
```

The function expects an association list of operators and functions in the *ops* argument, where each key is an operator symbol and each value is the name of a function implementing the operator. Examples will be given below.

The *parent-parser* argument will be bound to a function parsing the factors of the operators. Each factor of a chain of operations may contain operations of a higher precedence. These higher-prece-

zen style programming

dence operations are handled by *parent-parser*.

This function is a generalization of the *sum* and *term* functions of the *infix->prefix* parser introduced on page 113.

```
(define (parse-binary program ops parent-parser)
  (letrec
    ((lhs (parent-parser program))
     (collect
      (lambda (expr tlist)
        (let ((op (cond ((null tlist) :f)
                        (t (assq (car tlist) ops)))))
          (cond ((null tlist)
                 (make-prog expr ()))
                (op (let ((next (parent-parser
                                (make-prog () (cdr tlist)))))
                     (collect (list (cdr op) expr (s-expr-of next)
                                     (rest-of next))))
                  (t (make-prog expr tlist)))))))
    (collect (car lhs) (rest-of lhs))))
```

Parse-binary-r is like *parse-binary*, but parses and synthesizes right-associative operators:

$x \text{ op } y \text{ op } z \rightarrow (\text{fun}_{\text{op}} x (\text{fun}_{\text{op}} y z))$

It is a generalization of the *power* function of the *infix->prefix* parser [page 113].

```
(define (parse-binary-r program ops parent-parser)
  (let ((lhs (parent-parser program)))
    (let ((op (cond ((null (rest-of lhs)) :f)
                    (t (assq (first-of-rest lhs) ops)))))
      (cond ((null (rest-of lhs)) lhs)
            (op (let ((rhs (parse-binary-r
                            (make-prog () (rest-of-rest lhs))
                            ops
                            parent-parser)))
                 (list (list (cdr op) (s-expr-of lhs) (s-expr-of rhs))
                       (rest-of rhs))))
            (t lhs)))))
```

The following functions implement expression parsing as described in the section discussing infix to prefix conversion. Each function implements a production on top of *parse-binary* or *parse-binary-r*. For instance, *parse-concat* parses concatenation operators and performs the following transformations:

$a::b \rightarrow (\text{cons } a \text{ } b)$
 $a++b \rightarrow (\text{append } a \text{ } b)$

The factors (*a*,*b*) of these operations are recognized by *parse-factor*.

```
(define (parse-concat program)
  (parse-binary-r program
    '((:: . cons)
      (++) . append))
  parse-factor))

; x^y → (expt x y)

(define (parse-power program)
  (parse-binary-r program
    '((^ . expt))
    parse-concat))

; x*y → (* x y)
; x/y → (/ x y)
; x//y → (quotient x y)
; x\\y → (remainder x y)

(define (parse-term program)
  (parse-binary program
    '((* . *)
      (/ . /)
      (// . quotient)
      (\\ . remainder))
    parse-power))

; x+y → (+ x y)
; x-y → (- x y)

(define (parse-sum program)
  (parse-binary program
    '((+ . +)
      (- . -))
    parse-term))

; x=y → (= x y)
; x<>y → ((lambda (x y) (not (= x y))) x y)
; x<y → (< x y)
; x>y → (> x y)
; x<=y → (<= x y)
; x>=y → (>= x y)

(define (parse-pred program)
  (parse-binary program
    '((= . =)
      (<> . (lambda (x y) (not (= x y))))
      (< . <)
      (> . >)
      (<= . <=)
      (>= . >=))
    parse-sum))
```

zen style programming

```
; x/\y  →  (and x y)

(define (parse-conj program)
  (parse-binary program
    '(/\ . and))
    parse-pred))

; x\/y  →  (or x y)

(define (parse-disj program)
  (parse-binary program
    '(/\/ . or))
    parse-conj))
```

The *parse-expr* function is the front end of the expression parser. It converts a token list representing an M-expression into a program structure.

```
(define (parse-expr tlist)
  (parse-disj (make-prog () tlist)))
```

Internal-definition parses definitions that are part of `where` clauses. It returns a list containing the name and the body of the function being defined:

```
f[a1 ...] := expr  →  (f (lambda (a1 ...) expr))
```

The resulting list has the form of a local definition as used by **letrec**.

```
(define (internal-definition program)
  (let ((head (parse-expr (rest-of program))))
    (cond ((eq (first-of-rest head) ':=)
      (let ((term (parse-expr (rest-of-rest head)))
            (make-prog
              (list (car (s-expr-of head))
                    (make-lambda
                      (cdr (s-expr-of head))
                      (s-expr-of term)))
              (rest-of term))))
        (t (bottom ':= 'expected 'at (rest-of program))))))
```

The *parse-compound* function parses the `where` part of a compound definition:

```
where f[x] := expr  →  ((f (lambda (x) expr))
  and g[x] := expr      (g (lambda (x) expr))
  ...                    ...)
```

The resulting list can be used as an environment in **letrec**.

```
(define (parse-compound program)
  (letrec
    ((compound
      (lambda (prog def-list)
        (let ((defn (internal-definition (make-prog () prog))))
```

```
(cond ((eq (first-of-rest defn) 'and)
      (compound (rest-of-rest defn)
                 (cons (s-expr-of defn) def-list)))
      (t (make-prog
          (reverse (cons (s-expr-of defn) def-list))
          (rest-of defn))))))
(compound program ()))
```

Create a **letrec** expression out of an environment and a body:

```
(define (make-letrec env term)
  (list 'letrec env term))
```

Parse-definition parses simple definitions as well as compound definitions. It returns applications of **define**:

```
f[x] := expr      → (define (f x) expr)
f[x] := y         → (define (f x)
  where g[x] := z   (letrec ((g (lambda (x) z)))
                    y))
```

When *parse-definition* is called, the head of the definition already has been parsed, so *program* contains a partial program like this: `((f x) (:= expr))`. The reasons will be explained below.

```
(define (parse-definition program)
  (let ((term (parse-expr (rest-of-rest program))))
    (cond ((eq (first-of-rest term) 'where)
          (let ((compound (parse-compound (rest-of-rest term)))
                (make-prog
                 (list 'define
                     (s-expr-of program)
                     (make-letrec (s-expr-of compound)
                                   (s-expr-of term)))
                     (rest-of compound))))
          (t (make-prog (list 'define
                              (s-expr-of program)
                              (s-expr-of term))
                        (rest-of term))))))
```

Parse-program parses a full M-expression program. Each program is either an expression or a definition, which introduces a problem: both kinds of sentence may share a common prefix of indefinite length:

```
f [ a1 ... a∞ ]
f [ a1 ... a∞ ] := expr
```

So the parser would need infinite look-ahead to decide what kind of input is contained in the token stream.

The problem is solved by *assuming* that the input contains an expression and parsing that. After parsing an expression the next input token is either a definition operator (`:=`) or not. When it is

zen style programming

a definition operator, the partially translated program is passed to *parse-definition*, which uses the S-expression generated so far as the head of a definition.

So the ambiguity in the grammar is actually handled at the semantic level (by rewriting the meaning of an S-expression) rather than at the syntactic level. This is a common technique in hand-crafted compilers.

```
(define (parse-program tlist)
  (let ((program (parse-expr tlist)))
    (cond ((eq (first-of-rest program) ':=)
           (parse-definition program))
          (t program))))
```

M-expr-compile compiles an M-expression to an S-expression and returns it:

```
(mexpr-compile '(f[x] := x)) => (define (f x) x)
```

The “rest” part of the program structure returned by *parse-program* must be empty or a syntax error has occurred.

```
(define (mexpr-compile source)
  (let ((program (parse-program (tokenize source))))
    (cond ((end-of program)
           (car program))
          (t (bottom 'syntax 'error 'at (rest-of program))))))
```

M-expr-eval compiles and evaluates an M-expression:

```
(define (mexpr-eval source)
  (eval (mexpr-compile source)))
```

The version of MEXPRC discussed in this chapter reads and compiles M-expressions, but emits results as S-expressions. Can you write a front end to MEXPRC that translates its output back to M-expression form? E.g.:

```
(mexpr '(1/2 ^ 1/2)) => '(1/4)
(mexpr '(%a :: <<b,c>>)) => '(<< a, b, c >>)
...
```

The hack that MEXPRC uses to distinguish between function applications and function definitions introduces a bug. It allows non-symbols in the formal argument lists of functions:

```
(mexpr-compile '(f[a+b, 17] := expr)) => '(define (f #+ab '#17) expr)
```

How would you fix this bug?

10.3 example programs

M-expression (MEXPR) programs can be stored in files, just like *zenlisp* programs. MEXPR programs must require MEXPRC, and the source code should be placed in the file as an argument

of *mexpr-eval*, as the following example illustrates:

```
(require 'mexprc)
(mexpr-eval '(
  m_fac[x] := [x=0 -> 1: m_fac[x-1] * x]
))
```

MEXPR programs of this form can be loaded using **load**. The application of *mexpr-eval* compiles the code automatically when the file is loaded. So the above example can be used like this (given that the code is stored in the file `m_fac.l`):

```
; user input is in italics
(load m_fac)
=> :t
(mexpr-eval '(m_fac[17]))
=> '#355687428096000
```

10.3.1 append lists

This is an MEXPR implementation of the *append2* function introduced on page 22:

```
(require 'mexprc)
(mexpr-eval '(
  m_append[a,b] :=
    r_append(reverse[a], b]
  where
    r_append[a,b] :=
      [null[a]
       -> b:
        r_append[cdr[a], car[a]::b]]
))
```

Unfortunately, this function does not accept variable numbers of arguments. In the case of **append** this is not too bad, because

```
(append a b c d)
```

translates to

```
a ++ b ++ c ++ d
```

but variadic functions may be useful under other circumstances. Can you devise a notation for variadic MEXPR functions? Can you implement it by modifying MEXPRC?

zen style programming

10.3.2 the towers of hanoi

You *do* know the rules of the *Towers of Hanoi*, don't you? If not, here is a brief description:

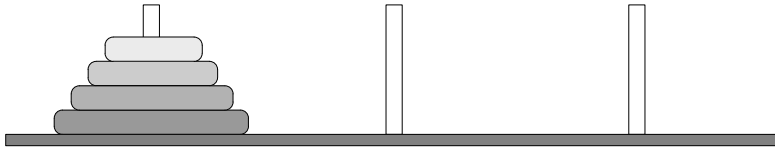


Fig. 8 – the towers of hanoi

There are three poles and a number of disks of different diameters as depicted in figure 8. Originally all disks are stacked on one pole in such a way that a smaller disk is always placed on top of a larger disk. The task is to move all disks to another pole — *but*:

- only one disk may be moved at a time;
- disks may only move from one pole to another;
- no larger disk may be placed on top of a smaller one.

Good luck!

Of course, being a programmer, you would not want to solve this puzzle in your head. You would want to find a *general* solution in the form of a program that finds the minimum set of moves required to solve the puzzle for a given number of disks.

Feel free to find such a solution before looking at the following MEXPR program. The solution is not too hard once you have understood how the puzzle is solved.

```
(require 'mexprc)

(mexpr-eval '(
  m_hanoi[n] :=
    solve[%LEFT, %MIDDLE, %RIGHT, n]
  where
    solve[from, to, via, n] :=
      [n=0
        -> nil:
        solve[from, via, to, n-1]
        ++ list[ list[from, to] ]
        ++ solve[via, to, from, n-1]]
))
```

What complexity does the *m_hanoi* program have? Can it be improved?

10.3.3 n queens

The rules of the *eight queens* puzzle are even simpler than those of the Towers of Hanoi: Place eight queens on a chess board in such a way that no queen may attack another. In case you do not play chess or prefer a less violent description: place eight objects in an 8×8 grid in such a way that no two objects can be connected by a horizontal, vertical, or diagonal line.

N queens is a generalization of this puzzle that uses an $n \times n$ grid instead of a chess board. The program of this section returns the first solution for a board of a given size, e.g.:

```
(mexpr-eval '(m_queens[4])) => ' (#1 #7 #8 #14)
```

Solutions are lists of fields. Fields are enumerated rather than giving them x/y coordinates. Above solution translates to the following grid (objects are placed on boldface numbers):

```
3      7    11   15
2      6    10   14
1      5      9   13
0      4      8   12
```

The problem is typically solved using *backtracking*. A piece is placed in the first row of each column and when it conflicts with a previously placed piece, the next row is tried. When there are no more rows in a column, the algorithm *backtracks* to the previous column and moves the piece in that column to the next row. The program finishes when a piece can be successfully placed in the last column or when it attempts to backtrack in the first column.

When a piece was placed in the last column, the algorithm found a solution. When it attempts to backtrack in the first column, no solution exists for the given grid size.

Obviously there is no solution for a 2×2 grid:

```
1 3      1 3      1 3      1 3
0 2      0 2      0 2      0 2
```

In the same way you can show that there is no solution for a 3×3 grid, but there already are 27 possible configurations to test. Larger grids are best checked by a program. Here it is:

```
(require 'mexprc)

(mexpr-eval '(
  m_queens[size] :=
    n_queens[0, 0, nil]

  where n_queens[q, c, b] :=
    [c = size
     -> reverse[b]:
```

zen style programming

```
column[q] <> c
-> [null[b]
    -> nil:
        n_queens[car[b]+1, c-1, cdr[b]]]:
safe_place[q, b]
-> n_queens[next_column[q], c+1, q::b]:
n_queens[q+1, c, b]]

and column[x] := x // size

and row[x] := x \\ size

and safe_place[x,b] :=
[null[b]
 -> true:
    connected[car[b], x]
 -> false:
    safe_place[x, cdr[b]]]

and connected[x,y] :=
common_h_v[x,y] \ / common_dia[x,y]

and common_h_v[x,y] :=
row[x] = row[y] \ / column[x] = column[y]

and common_dia[x,y] :=
abs[column[x]-column[y]] = abs[row[x]-row[y]]

and next_column[q] := [q+size] // size * size

))
```

This program prints only the first solution for a given grid size. Can you modify it to print all solutions?

What worst-case complexity does *m_queens* have? What average complexity does it have?

Do you think that solutions exist for all grid sizes greater than 3×3?

11. another micro kanren

The `amk` (*Another Micro Kanren*) package embeds declarative logic programming in `zenlisp`. It is based on ideas presented in “The Reasoned Schemer”²⁰ by Daniel P. Friedman, et al. The code is also inspired by the “Sokuza Mini-Kanren” implementation by Oleg Kiselyov.

`Amk` may be considered a language of its own. It is a logic programming language — like `PROLOG` — rather than a functional programming language. However, it integrates seamlessly into `zenlisp`: `zenlisp` data may be passed to `amk` and `amk` returns ordinary S-expressions as its results.

In order to use `amk` in `zenlisp` programs, the **`amk`** package must be loaded by beginning a program with the following expression:

```
(require '~amk)
```

11.1 introduction

11.1.1 functions versus goals

In functional programming, functions are combined to form programs. For example, the **`append`** function of `zenlisp` concatenates lists:

```
(append '#orange '#-juice) => '#orange-juice
```

The basic building stones of logic programming are called *goals*. A goal is a function that maps knowledge to knowledge:

```
(run* () (appendo '#orange '#-juice '#orange-juice)) => '(())
```

An application of **`run*`** is called a *query*. **`Run*`** is the interface between `zenlisp` and `amk`. The result of **`run*`** is called the *answer* to the corresponding query.

The goal used in the above query is **`append`**⁰ [page 177].

An answer of the form `'(())` means “success”. In the above example, this means that `'#orange-juice` is indeed equal to the concatenation of `'#orange` and `'#-juice`.

A goal returning a positive answer is said to *succeed*.

```
(run* () (appendo '#orange '#-juice '#fruit-salad)) => ()
```

does *not* succeed, because `'#fruit-salad` cannot be constructed by appending `'#orange` to `'#-juice`.

A goal that does not succeed is said to *fail*. Failure is represented by `()`.

²⁰ Friedmann, Byrd, Kiselyov; “The Reasoned Schemer”; MIT Press, 2005

zen style programming

When one or more arguments of a goal are replaced with variables, the goal attempts to *infer* the values of these variables.

Logic variables are created by the **var** function:

```
(define vq (var 'q))
```

Any argument of a goal can be a variable:

```
(run* vq (appendo '#orange '#-juice vq)) => '(#orange-juice)
(run* vq (appendo '#orange vq '#orange-juice)) => '(#-juice)
(run* vq (appendo vq '#-juice '#orange-juice)) => '(#orange)
```

In these sample queries, **run*** is told that we are interested in the value of *vq*. It runs the given goal and then returns the value or values of *vq* rather than just success or failure.

Goals are non-deterministic, so a query may return more than a single answer:

```
(run* vq (let ((dont-care (var 'dont-care)))
            (appendo dont-care vq '#abcd)))
=> '(#abcd #bcd #cd #d ())
```

This query returns all values which give '#abcd when appended to something that is not interesting. In other words, it returns all suffixes of '#abcd.

Subsequently, the following query returns all prefixes:

```
(run* vq (let ((dont-care (var 'dont-care)))
            (appendo vq dont-care '#abcd)))
=> '(() #a #ab #abc #abcd)
```

What do you think is the answer to the following query?

```
(run* vq (let ((x (var 'x))
                (y (var 'y)))
            (appendo x y vq)))
```

Does it have an answer at all?

Answer: The query has no answer, because there is an indefinite number of combinations that can be used to form a concatenation with an unspecific prefix and suffix. So **append**⁰ never stops generating combinations of values for its variables.

11.1.2 unification

Unification is an algorithm that forms the heart of every logic programming system. The “unify” goal is written **==**. The query

```
(run* vq (== x y))
```

means “unify *x* with *y*”. The answer of this query depends on the values of *x* and *y*:

```
(run* vq (== 'pizza 'pizza)) => '(() )
(run* vq (== 'cheese 'pizza)) => ()
(run* vq (== vq vq))          => '(() )
(run* vq (== 'cheese vq))     => '(cheese)
```

When two atoms are passed to `==`, it succeeds if the atoms are equal.

When a variable is passed to `==`, the variable is *bound* to the other argument:

```
(run* vq (== vq 'cheese)) => '(cheese)
(run* vq (== 'cheese vq)) => '(cheese)
```

The order of arguments does not matter.

When two variables are unified, these two variables are guaranteed to *always* bind to the same value:

```
(run* vq (let ((vx (var 'x)))
              (== vq vx)))
```

makes `vq` and `vx` bind to the same value. Binding a value to one of them at a later time automatically binds that value to both of them.

Non-atomic arguments are unified recursively by first unifying their car parts and then unifying their cdr parts:

```
(run* vq (== '(x (y) z) '(x (y) z))) => '(() )
(run* vq (== '(x (y) z) '(x (X) z))) => ()
(run* vq (== vq '(x (y) z)))          => '((x #y z))
```

Inference works even if variables are buried inside of lists:

```
(run* vq (== (list 'x vq 'z) '(x #y z))) => '(#y)
```

Because `'#y` is the only value for `vq` that makes the goal succeed, that value is bound to `vq`.

How does this work?

- `'x` is unified with `'x`;
- `vq` is unified with `'#y` (binding `vq` to `'#y`);
- `'z` is unified with `'z`.

Each unification may expand the “knowledge” of the system by binding a variable to a value or unifying two variables.

When unifying lists, the cdr parts of the lists are unified in the context of the knowledge gained during the unification of their car parts:

```
(run* vq (== '(      pizza fruit-salad)
              (list vq      vq              ))) => ()
```

This unification cannot succeed, because `vq` is first bound to `'pizza` and then the same variable

zen style programming

is unified with `'fruit-salad`.

When `vq` is unified with `'pizza`, `vq` is *fresh*. A variable is fresh if it is not (yet) bound to any value.

Only fresh variables can be bound to values.

When a form is unified with a bound variable, it is unified with the *value* of that variable. Hence

```
(run* vq (== '(pizza fruit-salad) (list vq vq))) => ()
```

is equivalent to

```
(run* vq (== '(pizza fruit-salad) (list vq 'pizza))) => ()
```

The following query succeeds because no contradiction is introduced:

```
(run* vq (== '(pizza pizza) (list vq vq))) => '(pizza)
```

First `vq` is unified with `'pizza` and then *the value of `vq`* (which is `'pizza` at this point) is unified with `'pizza`.

Bound variables can still be unified with fresh variables:

```
(run* vq (let ((vx (var 'x)))  
  (== (list 'pizza vq)  
      (list vx vx)))) => '(pizza)
```

Here `vx` is unified with `'pizza` and then the fresh variable `vq` is unified with `vx`, binding `vq` and `vx` to the same value.

Again, the order of unification does not matter:

```
(run* vq (let ((vx (var 'x)))  
  (== (list vq 'pizza)  
      (list vx vx)))) => '(pizza)
```

11.1.3 logic operators

The **any** goal succeeds, if at least one of its *subgoals* succeeds:

```
(run* vq (any (== vq 'pizza)  
              (== 'orange 'juice)  
              (== 'yes 'no)))  
=> '(pizza)
```

In this example, one of the three subgoals succeeds and contributes to the answer.

Because **any** succeeds if at least one of its subgoals succeeds, it fails if no subgoals are given:

```
(run* () (any)) => ()
```

Multiple subgoals of **any** may unify the same variable with different forms, giving a non-deterministic answer:

```
(run* vq (any (== vq 'apple)
              (== vq 'orange)
              (== vq 'banana)))
=> '(apple orange banana)
```

No contradiction is introduced. *Vq* is bound to each of the three values.

The **any** goal implements the *union* of the knowledge gained by running its subgoals:

```
(run* vq (any fail
              (== vq 'fruit-salad)
              fail))
=> '(fruit-salad)
```

It succeeds even if some of its subgoals fail. Therefore it is equivalent to the *logical or*.

Fail is a goal that always fails.

The **all** goal is a cousin of **any** that implements the *logical and*:

```
(run* vq (all (== vq 'apple)
              (== 'orange 'orange)
              succeed))
=> '(apple)
```

Succeed is a goal that always succeeds.

All succeeds only if all of its subgoals succeed, but it does more than this.

All forms the intersection of the knowledge gathered by running its subgoals by removing any contradictions from their answers:

```
(run* vq (all (== vq 'apple)
              (== vq 'orange)
              (== vq 'banana))) => ()
```

This query fails because *vq* cannot be bound to 'apple and 'orange and 'banana at the same time.

The effect of **all** is best illustrated in combination with **any**:

```
(run* vq (all (any (== vq 'orange)
                  (== vq 'pizza))
              (any (== vq 'apple)
                  (== vq 'orange))))
=> '(orange)
```

The first **any** binds *vq* to 'orange or 'pizza and the second one binds it to 'apple or 'orange.

zen style programming

All forms the intersection of this knowledge by removing the contradictions $vq = \text{'pizza}$ and $vq = \text{'apple}$. $Vq = \text{'orange}$ is no contradiction because it occurs in both subgoals of **all**.

BTW: **all** fails if at least one of its subgoals fails. Therefore it succeeds if no goals are passed to it:

```
(run* () (all)) => '(() )
```

11.1.4 parameterized goals

A parameterized goal is a function returning a goal:

```
(define (conso a d p) (== (cons a d) p))
```

Applications of **conso** evaluate to a goal, so **cons**⁰ can be used to form goals in queries:

```
(run* vq (conso 'heads 'tails vq)) => '((heads . tails))
```

In the prose, **conso** is written **cons**⁰. The trailing “o” of goal names is pronounced separately (e.g. “cons-oh”).

Obviously, **cons**⁰ implements something that is similar to the **cons** function.

However, **cons**⁰ can do more:

```
(run* vq (conso 'heads vq '(heads . tails))) => '(tails)
(run* vq (conso vq 'tails '(heads . tails))) => '(heads)
```

So **conso**⁰ can be used to define two other useful goals:

```
(define (caro p a) (conso a (__) p))
```

Car⁰ is similar to the **car** function of zenlisp and **cdr**⁰ is similar to its **cdr** function:

```
(define (cdro p d) (conso (__) d p))
```

Like the `_` variable of PROLOG, the expression `(__)` indicates a value that is of no interest.

When the second argument of **car**⁰ and **cdr**⁰ is a variable, they resemble **car** and **cdr**:

```
(run* vq (caro '(x . y) vq)) => '(x)
(run* vq (cdro '(x . y) vq)) => '(y)
```

Like **cons**⁰, **car**⁰ and **cdr**⁰ can do more than their zenlisp counterparts, though:

```
(run* vq (caro vq 'x)) => '((x . __, 0))
(run* vq (cdro vq 'y)) => '(__, 0 . y)
```

The query

```
(run* vq (caro vq 'x))
```

asks: “what has a car part of 'x?’” and the answer is “any pair that has a car part of 'x and a cdr part that does not matter.”

Clever, isn't it?

11.1.5 reification

Atoms of the form `_n`, where *n* is a unique number, occur whenever an answer would otherwise contain fresh variables:

```
(run* vq (let ((vx (var 'x))
               (vy (var 'y))
               (vz (var 'z)))
            (== vq (list vx vy vz))))
=> '(_0 _1 _2)
```

In the remainder of this document, `_n` may be spelled `_n`.

`_0`, `_1`, etc are called *reified variables*.

The replacement of fresh variables with reified names is called *reification*. It replaces each fresh variable with a unique “item” (*res* being the latin word for “item”).

11.1.6 recursion

Here is a recursive `zenlisp` predicate:

```
(define (mem-p x l)
  (cond ((null l) :f)
        ((eq x (car l)) :t)
        (t (mem-p x (cdr l)))))
```

Mem-p tests whether *l* contains *x*:

```
(mem-p 'c '#abcdef) => :t
(mem-p 'x '#abcdef) => :f
```

In `amk` you cannot write code like `(eq x (car l))` because

In logic programming, there is no function composition.
--

Each argument of a goal *must* be either a datum or a variable. Only **any** and **all** have subgoals:

```
(define (memo x l)
  (let ((va (var 'a))
        (vd (var 'd)))
    (any (all (caro l va)
              (eqo x va))
         (all (cdro l vd)
              (lambda (s)
                ((memo x vd) s))))))
```

zen style programming

Here are some observations:

- One **any** containing one or multiple **all** goals is the logic programming equivalent of **cond**;
- Each time **mem**⁰ is entered, a fresh *va* and *vd* is created;
- **Mem**⁰ does not seem to check whether *l* is **()**;
- The recursive case uses “eta conversion” to avoid early recursion (see below).

Does **mem**⁰ work? Yes:

```
(run* () (memo 'c '#abcdef)) => '()  
(run* () (memo 'x '#abcdef)) => ()
```

How does it work?

The first **all** unifies the car part of *l* with *va*. In case *l* = **()**, **all** fails.

Eq⁰ is a synonym for **==**.

If *va* (which is now an alias of **(car l)**) can be unified with *x*, this branch of **any** succeeds.

If *l* is **()**, both of these goals fail:

```
(caro l va) => ()  
(cdro l vd) => ()
```

and so the entire **mem**⁰ fails. There is no need to test for *l* = **()** explicitly.

The second **all** of **mem**⁰ unifies *vd* with the cdr part of *l* and then recurses.

Because **any** and **all** are ordinary **zenlisp** functions, recursion would occur *before* running **all**, if the application of **mem**⁰ was not protected by eta expansion.

Eta expansion wraps a lambda function around another function (and eta reduction removes that “wrapper”):

```
(memo x vd)  <----->  (lambda (s) ((memo x vd) s))  
                  eta
```

Both conversions maintain the meaning of the original function, except for the time of its reduction: **(memo x vd)** would be reduced immediately (delivering a goal) while in

```
(lambda (s) ((memo x vd) s))
```

the goal is only created when the enclosing lambda function is applied to a value. Eta expansion effectively implements “*call-by-name*” semantics.

All recursive cases must be protected using eta expansion.

11.1.7 converting predicates to goals

A predicate is a function returning a truth value.

Each goal is a predicate in the sense that it either fails or succeeds.

There are five steps involved in the conversion of a predicate to a goal:

- c1. Decompose function compositions;
- c2. Replace functions by parameterized goals;
- c3. Replace **cond** with **any** and its clauses with **all**;
- c4. Remove subgoals that make the predicate fail;
- c5. Protect recursive cases using eta expansion.

Mem-p [page 171] is converted as follows:

```
((eq x (car l)) :t)
```

becomes (by c1, c2, c3)

```
(let ((va (var 'a)))  
  (all (caro l va)  
        (eqo x va)))
```

and

```
(t (mem-p x (cdr l)))
```

becomes (by c1, c2, c3, c5)

```
(let ((vd (var 'd)))  
  (all (cdro l vd)  
        (lambda (s)  
          ((memo x vd) s))))
```

Finally

```
((null l) :f)
```

is removed (by c4) and **cond** is replaced with **any** (by c3).

In the original definition of **mem**⁰, (let ((va ...))) and (let ((vd ...))) are combined and moved outside of **any**.

BTW, the application of **eq**⁰ in **mem**⁰ is redundant, because **car**⁰ itself could make sure that *x* is the car part of *l*. So **mem**⁰ can be simplified significantly:

```
(define (memo x l)  
  (let ((vt (var 't)))  
    (any (caro l x)  
          (all (cdro l vt)  
                (lambda (s)  
                  ((memo x vt) s))))))
```

zen style programming

11.1.8 converting functions to goals

Memq is similar to **mem-p**:

```
(define (memq x l)
  (cond ((null l) :f)
        ((eq x (car l)) l)
        (t (memq x (cdr l)))))
```

Instead of returning just **:t** in case of success, it returns the first sublist of *l* whose car part is *x*:

```
(memq 'orange '(apple orange banana)) => '(orange banana)
```

Functions are converted to goals in the same way as predicates, but there is one additional rule:

c6 . Add an additional argument to unify with the result.

Memq⁰ is similar to **mem⁰**, but it has an additional argument *r* for the result and an additional goal which unifies the result with *r*:

```
(define (memqo x l r)
  (let ((vt (var 't)))
    (any (all (caro l x)
              (== l r))
         (all (cdro l vt)
              (lambda (s)
                ((memqo x vt r) s))))))
```

Like **memq**, **memq⁰** can be queried to deliver the first sublist of *l* whose head is *x*:

```
(run* vq (memqo 'orange '(apple orange banana) vq))
=> '((orange banana))
```

Memq⁰ even delivers *all* the sublists of *l* beginning with *x*:

```
(run* vq (memqo 'b '#abababc vq))
=> '(#bababc #babc #bc)
```

If you are only interested in the first one, take the car part of the answer.

Memq⁰ can be used to implement the identity function:

```
(run* vq (memqo vq '(orange juice) ( _)))
=> '(orange juice)
```

How does this work?

The question asked here is “what should *vq* be unified with to make `(memqo vq '(orange juice) (_))` succeed?”

The **car⁰** in **memq⁰** unifies *vq* (which is unified with *x*) with `'orange` and because *vq* is fresh, it succeeds.

The second case also succeeds. It binds l to $'(juice)$ and retries the goal. In this branch, vq is still fresh.

The car^0 in memq^0 unifies vq with $'juice$ and because vq is fresh, it succeeds.

The second case also succeeds. It binds l to $()$ and retries the goal. In this branch, vq is still fresh.

$(\text{memq } vq () ())$ fails, because neither car^0 nor cdr^0 can succeed with $l=()$.

Any forms the union of $vq='orange$ and $vq='juice$, which is the answer to the query.

11.1.9 cond versus any

LISP's **cond** pseudo function tests the predicates of its clauses sequentially and returns the normal form of the expression associated with the first true predicate:

```
(cond (t 'bread)
      (:f 'with)
      (t 'butter)) => 'bread
```

Even though the clause $(t 'butter)$ also has a true predicate, the above **cond** will *never* return $'butter$.

A combination of **any** and **all** can be used to form a logic programming equivalent of **cond**:

```
(run* vq (any (all succeed (== vq 'bread))
              (all fail (== vq 'with))
              (all succeed (== vq 'butter))))
=> '(bread butter)
```

Any replaces **cond** and **all** introduces each individual case.

Unlike **cond**, though, this construct returns the values of *all* cases that succeed.

While **cond** ignores the remaining clauses in case of success, **any** keeps trying until it runs out of subgoals.

This is the reason why memq^0 [page 174] returns all sublists starting with a given form:

```
(run* vq (memq 'b '#abababc vq)) => '(#bababc #babc #bc)
```

This is how it works:

When the head of l is *not* equal to $'b$, the first subgoal of **any** in memq^0 fails, so nothing is added to the answer.

When the head of l is equal to $'b$, the first subgoal of **any** succeeds, so l is added to the answer.

In either case, the second goal is tried. It succeeds as long as l can be decomposed. It fails when the end of the list l has been reached.

zen style programming

When the second goal succeeds, the whole **any** is tried on the cdr part of *l*, which may add more sublists to the answer.

What happens when the order of cases is reversed in `memq`⁰?

```
(define (rmemq x l r)
  (let ((vt (var 't)))
    (any (all (cdro l vt)
              (lambda (s)
                ((rmemq x vt r) s)))
         (all (caro l x)
              (== l r))))))
```

Because **any** keeps trying until it runs out of goals, **rmemq**⁰ does indeed return all matching sublists, just like **memq**⁰. However...

```
(run* vq (memq 'b '#abababc vq)) => '(#bababc #babc #bc)
(run* vq (rmemq 'b '#abababc vq)) => '(#bc #babc #bababc)
```

Because **rmemq**⁰ first recurses and then checks for a matching sublist, its answer lists the last matching sublist first.

Reversing the goals of **memq**⁰ makes it return its results in reverse order.

While **memq**⁰ can implement the identity function, **rmemq**⁰ can implement a function that reverses a list:

```
(run* vq (memq vq '(ice water) ( _))) => '(ice water)
(run* vq (rmemq vq '(ice water) ( _))) => '(water ice)
```

11.1.10 first class variables

Logic variables are first class values.

When a bound logic variable is used as an argument of a goal, the value of that variable is passed to the goal:

```
(run* vq (let ((vx (var 'vx)))
  (all (== vx 'piece-of-cake)
       (== vq vx))))
=> '(piece-of-cake)
```

When a fresh variable is used as an argument of a goal, the *variable itself* is passed to that goal:

```
(run* vq (let ((vx (var 'x)))
  (== vq vx)))
=> '(_ , 0)
```

Because the variable *vx* is fresh, it is reified by the interpreter after running the query, giving `_0`.

Variables can even be part of compound data structures:

```
(run* vq (let ((vx (var 'x)))
             (conso 'heads vx vq)))
=> '((heads . _ , 0))
```

Unifying a variable that is part of a data structure at a later time causes the variable part of the data structure to be “filled in” belatedly:

```
(run* vq (let ((vx (var 'x)))
             (all (conso 'heads vx vq)
                  (== vx 'tails))))
=> '((heads . tails))
```

The **append**⁰ goal makes use of this fact:

```
(define (appendo x y r)
  (any (all (== x ())
            (== y r))
       (let ((vh (var 'h))
             (vt (var 't))
             (vtr (var 'tr)))
         (all (conso vh vt x)
              (conso vh vtr r)
              (lambda (s)
                ((appendo vt y vtr) s)))))))
```

How is the following query processed?

```
(run* vq (appendo '#ab '#cd vq))
```

In its recursive case, **append**⁰ first decomposes $x = \text{'\#ab}$ into its head $vh = \text{'a}$ and tail $vt = \text{'\#b}$:

```
(conso vh vt x)
```

The next subgoal states that the head vh consed to vtr (the tail of the result) gives the result of **append**⁰:

```
(conso vh vtr r)
```

Because vtr is fresh at this point, r is bound to a structure containing a variable:

```
r0 = (cons 'a vtr0)
```

Vtr and r are called vtr_0 and r_0 here, because they are the first instances of these variables.

When the goal recurses, vtr_0 is passed to **append**⁰ in the place of r :

```
(appendo '#b '#cd vtr0)
```

Append⁰ creates fresh instances of vtr and r (called vtr_1 and r_1).

At this point r_1 and vtr_0 may be considered *the same variable*, so

zen style programming

```
(conso vh vtr1 r1)
```

results in

```
r1 = vtr0 = (cons 'b vtr1)
```

and

```
r0 = (cons 'a vtr0) = (cons 'a (cons 'b vtr1))
```

When **append**⁰ recurses one last time, *vtr₁* is passed to the goal in the place of *r* and the instance *r₂* is created:

```
(appendo () '#cd vtr1)
```

Because *x=()*, the subgoal handling the trivial case is run:

```
(== y r2)
```

And because *r₂* and *vtr₁* are the same variable,

```
r2 = vtr1 = '#cd  
r1 = vtr0 = (cons 'b vtr1)  
           = (cons 'b '#cd)  
r0 = (cons 'a vtr0)  
     = (cons 'a (cons 'b vtr1))  
     = (cons 'a (cons 'b '#cd))
```

11.1.11 first class goals

Like LISP functions, goals are first class values.

The *filter*⁰ goal makes use of this fact:

```
(define (filtero p l r)  
  (let ((va (var 'a))  
        (vd (var 'd)))  
    (any (all (caro l va)  
              (p va)  
              (== va r))  
          (all (cdro l vd)  
                (lambda (s)  
                  ((filtero p vd r) s))))))
```

*Filter*⁰ extracts all members with a given property from a list.

The property is described by the goal *p* which is passed as an argument to *filter*⁰:

```
(run* vq (filtero paio '(a b (c . d) e (f . g)) vq))  
=> '((c . d) (f . g))
```

Pair⁰ is defined this way:

```
(define (paio x) (conso (_) (_) x))
```

Because parameterized goals are ordinary functions, though, there is no need to invent a new function name. **Lambda** works fine:

```
(run* vq (filtero (lambda (x) (conso (_) (_) x))  
                  '(a b (c . d) e (f . g)) vq))  
=> '((c . d) (f . g))
```

11.1.12 negation

The **neg** goal succeeds if its subgoal fails, and fails if its subgoal succeeds:

```
(run* () (neg fail)) => '()  
(run* () (neg succeed)) => ()
```

Neg *never* contributes any knowledge:

When its subgoal succeeds, **neg** itself fails, thereby deleting all knowledge gathered by its subgoal.

When its subgoal fails, there is no knowledge to add.

However, **neg** is not as straight-forward as it seems to be:

```
(define (nullo x) (eqo () x))  
(run* vq (neg (nullo vq)))
```

The **null**⁰ goal tests whether its argument is **()**.

What should be the answer to the question “what is not equal to **()**?”

Neg answers this question using an approach called the “closed world assumption”, which says “what cannot be proven true must be false”.

So the answer to above question is “nothing”. Because the value of *vq* is not known, **neg** cannot prove that it is not equal to **()** and fails:

```
(run* vq (neg (nullo vq))) => ()
```

Technically, it works like this:

Vq is fresh, so **null**⁰ unifies it with **()** and succeeds. Because **null**⁰ succeeds, **neg** must fail.

This approach has its consequences:

zen style programming

```
(run* vq
  (all (any (== vq 'orange)
            (== vq 'pizza)
            (== vq 'ice-cream))
        (neg (== vq 'pizza))))
=> '(orange ice-cream)
```

```
(run* vq
  (all (neg (== vq 'pizza))
        (any (== vq 'orange)
              (== vq 'pizza)
              (== vq 'ice-cream)))))
=> ()
```

Depending on its context, **neg** has different functions.

In the right example it makes the entire query fail, because the fresh variable *vq* can be unified with 'pizza.

In the left example, where *vq* already has some values, it eliminates the unification of *vq* and 'pizza.

Therefore

Negation should be used with great care.
--

11.1.13 cutting

The **memq**⁰ goal [page 174] returned all sublists whose heads matched a given form:

```
(run* vq (memqo 'b '#abababc vq)) => '(#babababc #babc #bc)
```

For the case that you are *really, really* only interested in the first match, there is a technique called *cutting*.

It is implemented by the **one** goal:

```
(run* vq (one fail
              (== vq 'apple)
              (== vq 'pie)))
=> '(apple)
```

As soon as one subgoal of **one** succeeds, **one** itself succeeds immediately and “cuts off” the remaining subgoals.

The name of **one** suggests that at most **one** of its subgoals can succeed.

Using **one**, a variant of **memq**⁰ can be implemented which succeeds with the first match:

```
(define (firsto x l r)
  (let ((vd (var 'd)))
    (one (all (caro l x)
              (== r l))
         (all (cdro l vd)
              (lambda (s)
                ((firsto x vd r) s))))))
```

The only difference between **memq**⁰ and **first**⁰ is that **first**⁰ uses **one** in the place of **any**.

First⁰ cuts off the recursive case as soon as the first case succeeds:

```
(run* vq (firsto 'b '#abababc vq)) => ' (#bababc)
```

So **one** is much more like **cond** than **any**.

However, **one** supresses *backtracking*, which is one of the most interesting properties of logic programming systems.

Here is another predicate:

```
(define (juiceo x)
  (let ((vt (var 't)))
    (all (cdro x vt)
         (caro vt 'juice))))
```

Juice⁰ succeeds, if its argument is a list whose second element is equal to 'juice, e.g.:

```
(run* () (juiceo '(orange juice))) => ' (())
(run* () (juiceo '(cherry juice))) => ' (())
(run* () (juiceo '(apply pie ))) => ()
```

Given the **juice**⁰ predicate, **memq**⁰ can be used to locate your favorite juice on a menu:

```
(define menu '(apple pie    orange pie    cherry pie
               apple juice  orange juice  cherry juice))
(run* vq (all (memqo 'orange menu vq)
              (juiceo vq)))
=> ' ((orange juice cherry juice))
```

When **memq**⁰ finds the sublist starting with the 'orange right before 'pie, **juice**⁰ fails and backtracking is initiated.

Memq⁰ then locates the next occurrence of 'orange and this time **juice**⁰ succeeds.

Using **first**⁰ supresses backtracking and so our favorite juice is never found:

```
(run* vq (all (firsto 'orange menu vq)
              (juiceo vq)))
=> ()
```

Therefore

Cutting should be used with great care.

11.3 a logic puzzle

The *Zebra Puzzle* is a well-known logic puzzle.

It is defined as follows:

- Five persons of different nationality live in five houses in a row.
- The houses are painted in different colors.
- The persons enjoy different drinks and brands of cigarettes.
- All persons own different pets.
- The Englishman lives in the red house.
- The Spaniard owns a dog.
- Coffee is drunk in the green house.
- The Ukrainian drinks tea.
- The green house is directly to the right of the ivory house.
- The Old Gold smoker owns snails.
- Kools are being smoked in the yellow house.
- Milk is drunk in the middle house.
- The Norwegian lives in the first house on the left.
- The Chesterfield smoker lives next to the fox owner.
- Kools are smoked in the house next to the house where the horse is kept.
- The Lucky Strike smoker drinks orange juice.
- The Japanese smokes Parliaments.
- The Norwegian lives next to the blue house.

Who owns the zebra?

To solve the puzzle, two questions have to be answered:

- How to represent the data?
- How to add facts?

Five attributes are linked to each house, so the row of houses can be represented by a list of 5-tuples like this:

(nation cigarette drink pet color)

Known facts are represented by symbols and unknown ones by variables.

The fact “the Spaniard owns a dog” would look like this:

```
(list 'spaniard (var 'cigarette) (var 'drink) 'dog (var 'color))
```

The addition of facts is explained by means of a simpler variant of the puzzle with only two attributes and two houses:

```
(list (list (var 'person) (var 'drink))  
      (list (var 'person) (var 'drink)))
```

- In one house lives a Swede.
- In one house lives a tea drinker.
- In one house lives a Japanese who drinks coffee.
- The tea drinker lives in the left house.

Applying the first fact yields the following options (variables are rendered in *italics*):

```
'( ((Swede drink) house)  
    (house (Swede drink)) )
```

This means that the Swede (whose drink is unknown) can live in the first or in the second house.

Adding the second fact yields more options:

```
'( ((Swede Tea) house)  
    ((Swede drink) (person Tea))  
    ((person Tea) (Swede drink))  
    (house (Swede Tea)) )
```

The key to the application of facts is unification. The fact

```
(list 'Swede (var 'drink))
```

can be unified with any fresh variable like *h* (as in *house*):

```
(define h (var 'h))  
  
(run* h (== h (list 'Swede (var 'drink))))  
=> '((swede _,0))
```

To create *all* possible outcomes, the fact must be applied to *each* of the two houses:

```
(define fact (list 'Swede (var 'drink)))  
  
(run* h (let ((h1 (var 'house1))  
              (h2 (var 'house2)))  
          (all (== h (list h1 h2))  
              (any (== fact h1)  
                  (== fact h2)))))  
=> '(((swede _,0) _,1)  
    (__,0 (swede _,1)))
```

zen style programming

Remember: reified variables like `_0` and `_1` denote something that is not known and/or of no interest.

In the above answer, `_0` represents an unknown drink in the first outcome and an unknown house in the second one. `_1` represents an unknown house in the first outcome and an unknown drink in the second one.

Each new fact must be unified with all outcomes produced so far.

A goal which automatically unifies a fact with all outcomes found so far would be helpful. The `mem0` goal, which was defined earlier in this chapter [page 173], can do this.

`Mem0` tries to unify a given form with each member of a list. Replace “form” with “fact” and “list” with “outcome” and here we go:

```
(run* h (all (== h (list (var 'h1)
                          (var 'h2)))
             (memo (list 'Swede (var 'drink)) h)
             (memo (list (var 'person) 'Tea) h)))
=> '(((swede tea) _0)
      ((swede _0) (_1 tea))
      ((_0 tea) (swede _1))
      (_0 (swede tea)))
```

At this point the query is *underspecified*; the known facts are not sufficient to tell where the Swede lives or whether he drinks tea or not.

By adding the third fact, some outcomes are eliminated:

```
(run* h (all (== h (list (var 'house1)
                          (var 'house2)))
             (memo (list 'Swede (var 'drink)) h)
             (memo (list (var 'person) 'Tea) h)
             (memo (list 'Japanese 'Coffee) h)))
=> '(((swede tea) (japanese coffee))
      ((japanese coffee) (swede tea)))
```

The query is still underspecified, but because the third fact contradicts the assumption that the other person drinks tea, we now know that the Swede drinks tea. We also know that the other person is a Japanese and drinks coffee.

To add the final fact, another goal is needed. `Left0` checks whether *x* is directly on the left of *y* in the list *l*:

```
(define (leftto x y l)
  (let ((vt (var 't)))
    (any (all (caro l x)
              (cdro l vt)
              (caro vt y))
```

```
(all (cdro 1 vt)
      (lambda (s)
        ((lefto x y vt) s))))))
```

Using *left⁰*, the puzzle can be solved:

```
(run* h (all (== h (list (var 'h1)
                          (var 'h2)))
              (memo (list 'Swede (var 'drink)) h)
              (memo (list (var 'person) 'Tea) h)
              (memo (list 'Japanese 'Coffee) h)
              (lefto (list (var 'person) 'Tea)
                     (var 'house) h))))
=> '(((swede tea) (japanese coffee)))
```

To solve the Zebra Puzzle, another predicate is needed. It expresses that *x* is *next to y*.

X is next to *y*, if *x* is on the left of *y* or *y* is on the left of *x*, so:

```
(define (nexto x y l)
  (any (lefto x y l)
        (lefto y x l)))
```

Predicates expressing the position of a house in the row are not required, because houses can be placed directly in the initial record:

```
(list (list 'norwegian (var 'c1) (var 'd1) (var 'p1) (var 'o1))
      (var 'h2)
      (list (var 'n2) (var 'c2) 'milk (var 'p2) (var 'o2))
      (var 'h4)
      (var 'h5))
```

Having to invent lots of unique variable names for unknown parameters is a bit cumbersome, but because the initial values of these variables are not really interesting, they can be replaced with anonymous variables:

```
(list (list 'norwegian (_) (_) (_) (_))
      (_)
      (list (_) (_) 'milk (_) (_))
      (_)
      (_))
```

Here is the full code for solving the Zebra Puzzle:

```
(define (zebra)
  (let ((h (var 'h)))
    (run* h (all (== h (list (list 'norwegian (_) (_) (_) (_))
                              (_)
                              (list (_) (_) 'milk (_) (_))
                              (_)
                              (_)))
                  (memo (list 'englishman (_) (_) (_) 'red) h))
```


zen style programming

```
(lefto (list (_) (_) (_) (_) 'green)
        (list (_) (_) (_) (_) 'ivory) h)
(nexto (list 'norwegian (_) (_) (_) (_))
        (list (_) (_) (_) (_) 'blue) h)
(memo (list (_) 'kools (_) (_) 'yellow) h)
(memo (list 'spaniard (_) (_) 'dog (_) ) h)
(memo (list (_) (_) 'coffee (_) 'green) h)
(memo (list 'ukrainian (_) 'tea (_) (_)) h)
(memo (list (_) 'luckystrikes 'orangejuice (_) (_)) h)
(memo (list 'japanese 'parliaments (_) (_) (_)) h)
(memo (list (_) 'oldgoldts (_) 'snails (_) ) h)
(nexto (list (_) (_) (_) 'horse (_))
        (list (_) 'kools (_) (_) (_)) h)
(nexto (list (_) (_) (_) 'fox (_))
        (list (_) 'chesterfields (_) (_) (_)) h)
;
(memo (list (_) (_) 'water (_) (_)) h)
(memo (list (_) (_) (_) 'zebra (_) ) h))))
```

The program should be run with a node pool size of 1024K bytes (i.e. start the `zenlisp` interpreter using the command: `zl -n 1024K`).

```
(zebra) => '(((norwegian kools _,0 fox yellow)
              (ukrainian chesterfields tea horse blue)
              (englishman oldgoldts milk snails red)
              (japanese parliaments coffee zebra green)
              (spaniard luckystrikes orangejuice dog ivory)))
```

Note that the puzzle is in fact underspecified. The drink of the Norwegian is not known. In case you prefer a fully specified query, just uncomment the fact listing “water” as a drink in the above code.

11.3 implementation

The complete implementation is written in purely symbolic LISP.

The only library function that is needed is **length**. It is imported from `zenlisp`'s `nmath` package.

```
(define amk :t)

(require '~nmath)
```

11.3.1 basics

These are the **fail** and **succeed** goals. They are guaranteed to have disjoint results:

```
(define (fail x) ())

(define (succeed x) (list x))
```

Var creates a logic variable and *var-p* checks whether an object is a logic variable. Logic variables are represented by forms like $(? \ . \ x)$ where x is the name of the variable. Note that all logic variables are unique instances. [See page 55]

```
(define (var x) (cons '? x))

(define (var-p x)
  (and (not (atom x))
        (eq (car x) '?)))
```

Because all variables are unique, **var** can be used to create anonymous variables:

```
(define (_) (var '_))
```

Empty-s represents ignorance:

```
(define empty-s ())
```

Knowledge is represented by *substitutions*. Substitutions are implemented using association lists. *Empty-s* is an empty substitution.

Ext-s adds the association of the variable x with the value v to the substitution s .

```
(define (ext-s x v s) (cons (cons x v) s))
```

Walk looks up the value of x in the substitution s :

```
(walk vx '((vx . bread))) => 'bread
```

(Vx denotes the logic variable x , i.e. the form $(? \ . \ x)$.)

Walk may look like **assoc** or **assq**, but it does more:

```
(define (walk x s)
  (cond ((not (var-p x)) x)
        (t (let ((v (assq x s)))
              (cond (v (walk (cdr v) s))
                    (t x))))))
```

When the value associated with a variable is another variable, *walk* looks up that other variable, thereby following chains of variables:

```
(walk vx '((vx . vy) (vz . sushi) (vy . vz))) => 'sushi
```

This is how the unification of variables is implemented.

When the variable passed to *walk* is fresh or a fresh variable is found while following a chain of variables, the fresh variable is returned:

```
(walk vx empty-s) => vx
```

This is why fresh variables are first-class objects.

zenstyle programming

The *atomic* predicate is used instead of *atom*, because zenlisp implements closures as lists.

```
(define (atomic x)
  (or (atom x)
      (eq (car x) 'closure)))
```

Unify is the heart of amk. It unifies *x* with *y*, looking up values of variables in *s*.

```
(define (unify x y s)
  (let ((x (walk x s))
        (y (walk y s)))
    (cond ((eq x y) s)
          ((var-p x) (ext-s x y s))
          ((var-p y) (ext-s y x s))
          ((or (atomic x) (atomic y)) :f)
          (t (let ((s (unify (car x) (car y) s)))
                (and s (unify (cdr x) (cdr y) s)))))))
```

Upon success *unify* returns *s* or an extension of *s* with new substitutions added. In case of failure it returns **:f**.

11.3.2 goals

This is the **==** goal. **==** is like *unify*, but it succeeds or fails rather than returning a substitution or falsity.

```
(define (== g1 g2)
  (lambda (s)
    (let ((s2 (unify g1 g2 s)))
      (cond (s2 (succeed s2))
            (t (fail s))))))
```

Note that **==** returns a lambda function that must be applied to a substitution to let the unification take place:

```
(== vq 'orange-juice) => {closure #s}
((== vq 'orange-juice) empty-s) => '(((vq . orange-juice)))
```

Also note that when **==** succeeds, it adds another list around the resulting substitution.

Here is the **any** goal:

```
(define (any . g*)
  (lambda (s)
    (letrec
      ((try (lambda (g*)
               (cond ((null g*) (fail s))
                     (t (append ((car g*) s)
                                (apply try (cdr g*)))))
            (apply try g*))))))
```

It forms a *list of substitutions* by applying each member of the list of goals g^* to the given knowledge s and appending the results:

```
((any (== vq 'ice) (== vq 'cream)) empty-s)
=> '(((vq . ice)) ((vq . cream)))
```

In the list of substitutions returned by **any**, each individual substitution is free of conflicting associations.

Any is the only goal that may produce multiple substitutions.

Here is **all**:

```
(define (all . g*)
  (lambda (s)
    (letrec
      ((try (lambda (g* s*)
              (cond ((null g*) s*)
                    (t (try (cdr g*)
                           (apply append
                                (map (car g*) s*)))))
            (try g* (succeed s))))))
```

All applies its subgoals to the knowledge s^* .

Because some of its subgoals may be **any**s (which may produce multiple substitutions), **all** maps each goal over a list of substitutions.

The mapping eliminates all associations from all substitutions of s^* that contradict the current goal. The remaining results are appended to form a new list of substitutions s^*_1 .

The next subgoal of **all** is mapped over s^*_1 , giving s^*_2 , etc.

Each subgoal is applied to the conjunction of the subgoals applied so far.

One is similar to **any**, but instead of appending substitutions, it returns the result of the first subgoal that succeeds:

```
(define failed null)

(define (one . g*)
  (lambda (s)
    (letrec
      ((try (lambda (g*
              (cond ((null g*) (fail s))
                    (t (let ((out ((car g*) s)))
                        (cond ((failed out)
                              (apply try (cdr g*)))
                              (t out)))))))
            (apply try g*))))
```

zenstyle programming

Here is the **neg** goal. Its implementation is more straight-forward than its application:

```
(define (neg g)
  (lambda (s)
    (let ((out (g s)))
      (cond ((failed out) (succeed s))
            (t (fail s))))))
```

Choice is a utility goal that facilitates the creation of sets:

```
(run* vq (choice vq '(orange banana kiwi)))
```

is an abbreviation of

```
(run* vq (any (== vq 'orange)
              (== vq 'banana)
              (== vq 'kiwi)))
```

Here is the code:

```
(define (choice x lst)
  (cond ((null lst) fail)
        (t (any (== x (car lst))
                 (choice x (cdr lst))))))
```

11.3.2 interface

Occurs and *circular* are helper functions that will be used by *walk**, which is explained right after them.

Occurs checks whether the symbol or variable *x* occurs in the form *y*. Like *walk*, *occurs* follows chains of variables. Values of variables are looked up in *s*.

```
(define (occurs x y s)
  (let ((v (walk y s)))
    (cond ((var-p y) (eq x y))
          ((var-p v) (eq x v))
          ((atomic v) :f)
          (t (or (occurs x (car v) s)
                  (occurs x (cdr v) s))))))
```

A value of a variable that contains references to that variable is said to be *circular*:

```
((= vq (list vq)) empty-s) => '(((vq . (vq))))
```

A circular answer is not valid because it is self-referential.

The *circular* function checks whether the value of a variable is circular:

```
(define (circular x s)
  (let ((v (walk x s)))
```

```
(cond ((eq x v) :f)
      (t (occurs x (walk x s) s))))
```

*Walk** is like *walk*: it turns a variable *x* into a value *v*. In addition it replaces all variables found in *v* with their values.

*Walk** makes the answers computed by *cmk* comprehensible:

```
((all (== vq (list vx)) (== vx 'foo)) empty-s)
=> '((vx . foo) (vq . (vx)))
(walk* vq '((vx . foo) (vq . (vx))))
=> '(foo)
```

When *walk** encounters a fresh variable, it leaves it in the result.

When the variable to be *walk**ed is bound to a circular value, *walk** returns *:bottom*.

```
(define :bottom (var ' :bottom))

(define (walk* x s)
  (letrec
    ((w* (lambda (x s)
          (let ((x (walk x s)))
            (cond ((var-p x) x)
                  ((atomic x) x)
                  (t (cons (w* (car x) s)
                           (w* (cdr x) s)))))))
    (cond ((circular x s) :bottom)
          ((eq x (walk x s)) empty-s)
          (t (w* x s)))))
```

Reify-name generates a reified name.

```
(define (reify-name n)
  (implode (append '#_, n)))
```

Reify creates a substitution in which each fresh variable contained in the form *v* is associated with a unique reified name:

```
(reify (list vx vy vz)) => '((vz . _,2) (vy . _,1) (vx . _,0))
```

The value *v* that is passed to *reify* must have been *walk**ed before.

```
(define (reify v)
  (letrec
    ((reify-s
      (lambda (v s)
        (let ((v (walk v s)))
          (cond ((var-p v)
                 (ext-s v (reify-name (length s)) s))
                ((atomic v) s)))))
```

zen style programming

```
(t (reify-s (cdr v)
            (reify-s (car v)
                      s))))))
(reify-s v empty-s)))
```

Preserve-bottom implements bottom preservation. No surprise here.

Explanation: The term *bottom* is used in mathematics to denote an undefined result, like a diverging function. *Bottom preservation* is a principle that says that any form that contains a bottom element is itself equal to bottom.

When an answer contains the variable *:bottom*, that answer has a circular structure, and the query that resulted in that answer should fail.

```
(define (preserve-bottom s)
  (cond ((occurs :bottom s) ())
        (t s)))
```

Run* is the principal interface for submitting queries to *amk*:

```
(define (run* x g)
  (preserve-bottom
   (map (lambda (s)
          (walk* x (append s (reify (walk* x s)))))
        (g empty-s)))))
```

X may be a logic variable or *()*.

When *x* is a variable, **run*** returns the value or values of that variable. When *x*=*()*, it returns either *'()* or *()*.

When a query fails, **run*** returns *()*.

Run* runs the goal *g* and then *walk**s each substitution of the answer.

It also reifies the fresh variables contained in each resulting substitution.

utilities

These are some predefined variables that were assumed to be defined in the previous sections.

```
(define vp (var 'p))
(define vq (var 'q))
```

The following code contains a collection of utility goals that belong to *amk*. They were discussed in detail earlier in this chapter.

```
(define (conso a d p) (= (cons a d) p))

(define (caro p a) (conso a (_) p))
```

```
(define (cdro p d) (conso (d) p))

(define (pairo p) (conso (d) (d) p))

(define (eqo x y) (== x y))

(define (nullo a) (eqo a ()))

(define (memo x l)
  (let ((vt (var 't)))
    (any (caro l x)
          (all (cdro l vt)
                (lambda (s)
                  ((memo x vt) s)))))))

(define (appendo x y r)
  (any (all (== x (d))
            (== y r))
        (let ((vh (var 'h))
              (vt (var 't))
              (va (var 'a)))
          (all (conso vh vt x)
                (conso vh va r)
                (lambda (s)
                  ((appendo vt y va) s)))))))

(define (memqo x l r)
  (let ((vt (var 't)))
    (any (all (caro l x)
              (== l r))
          (all (cdro l vt)
                (lambda (s)
                  ((memqo x vt r) s)))))))

(define (rmemqo x l r)
  (let ((vt (var 't)))
    (any (all (cdro l vt)
              (lambda (s)
                ((rmemqo x vt r) s)))
          (all (caro l x)
                (== l r)))))
```


zen style programming

part three zenlisp implementation

Zenlisp is an interpreter for the purely symbolic LISP language introduced and discussed in the earlier parts of this book. It is written in ANSI C (C89) and zenlisp. This part reproduces the complete source code of the zenlisp interpreter including lots of annotations. The first chapter will describe the about 3000 lines of C code and the second one the about 1000 lines of zenlisp code which together constitute the complete system.

12. c part

The complete C part of the source code is contained in the file `z1.c`.

```
/*
 * zenlisp -- an interpreter for symbolic LISP
 * By Nils M Holm <nmh@t3x.org>, 2007, 2008
 * Feel free to copy, share, and modify this program.
 * See the file LICENSE for details.
 */
```

Zenlisp is a tree-walking interpreter that implements shallow binding, a constant-space mark and sweep garbage collector, and bignum arithmetics (although these are considered a rather esoteric feature and are contained in the LISP part entirely.)

12.1 prelude and data declarations

```
#include <stdlib.h>
#ifdef __TURBOC__
    #include <io.h>
    #include <alloc.h>
#else
    #include <unistd.h>
    #ifndef __MINGW32__
        #ifndef __CYGWIN__
            #define setmode(fd, mode)
        #endif
    #endif
#endif
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <fcntl.h>

#define VERSION 2
#define RELEASE "2008-09-19"
```

DEFAULT_NODES is the default size of the **node pool** used by zenlisp. The node pool has a

zen style programming

static size and cannot grow at run time. A different node size can be specified using a command line option. Larger node pools (up to some limit) typically mean faster operation and a bigger memory footprint.

Each *node* consists of “car” field, a “cdr” field, and a “tag” field. The first two have a size of an `int` each and the tag has a size of one `char`, so the total size of the node pool is computed as follows:

```
SizePool = Nodes * (2 * sizeof(int) + 1)
```

The `MINIMUM_NODES` constant specifies the size of the smallest pool that can hold the LISP part of the system.

```
#define DEFAULT_NODES    131072
#define MINIMUM_NODES    12280
```

`DEFAULT_IMAGE` is the location of the LISP image file to load when the system starts up. A different image file can be specified on the command line.

```
#ifndef DEFAULT_IMAGE
#define DEFAULT_IMAGE    "/u/share/zenlisp/zenlisp"
#endif
```

The `counter` structure is a portable mechanism for representing big numbers. It is used to gather data regarding allocation and reduction cycles at run time. Its `n` member stores ones, the `nlk` member thousands, etc.

```
struct counter {
    int    n, nlk, nlm, nlg;
};
```

The `Error_context` structure is used to store the context in which an error occurred, so the error may be reported at a later time. Each error message has the form

```
* file: line: function: message: expression
* additional argument
* Trace: function ...
```

The *file*, *expression*, *additional argument*, and *Trace: ...* parts are optional. The error context holds the individual parts of the message:

msg error message
arg additional argument
expr expression that caused the error or `NO_EXPR`
file input file or `NULL` for `stdin`
line input line number
fun function in which the error occurred or `NIL`
frame current call frame, used to print trace

```
struct Error_context {
    char    *msg;
    char    *arg;
    int     expr;
    char    *file;
    int     line;
    int     fun;
    int     frame;
};
```

This is the maximum length of a symbol name, and the maximum length of a source path:

```
#define SYMBOL_LEN      256
#define MAX_PATH_LEN    256
```

The following flags are used to control the garbage collector. `ATOM_FLAG` is also used to distinguish atoms from pairs. Seriously, there are no types in `zenlisp` other than the atom (symbol) and the pair.

ATOM_FLAG This node is an atom

MARK_FLAG Used to tag live nodes during GC

SWAP_FLAG Used to indicate that this node is not yet completely visited

When `SWAP_FLAG` is set, the “car” and “cdr” fields of the node will be swapped in order to visit the cdr child in garbage collections. Hence the name of this flag.

```
#define ATOM_FLAG      0x01
#define MARK_FLAG      0x02
#define SWAP_FLAG      0x04
```

Here are some magic values. `NIL` is the empty list. It spells out “not in list” and therefore its (internal) value is an invalid index of the node pool. `EOT` is the end of (input) text. It is used to denote the end of a file or TTY input. `DOT` and `R_PAREN` are delivered by the reader when a dot (“.”) or a right parenthesis (“)”) is found. `NO_EXPR` indicates that an error was not caused by any specific expression.

```
#define NIL      -1
#define EOT      -2
#define DOT      -3
#define R_PAREN -4
#define NO_EXPR  -5
```

These are the states that the `zenlisp` evaluator may go through while reducing a program to its normal form. They will be explained in detail in the description and code of the `eval` function.

MATOM evaluating an atom; this is the original state

MLIST evaluating a list of function arguments

MBETA evaluating the body of a function

MBIND evaluating the bindings of **let**

zen style programming

MBINR evaluating the bindings of **letrec**
MLETR evaluating the term of **let** or **letrec**
MCOND evaluating predicates of **cond**
MCONJ evaluating expressions of **and** (but not the last one)
MDISJ evaluating expressions of **or** (but not the last one)

```
enum Evaluator_states {
    MATOM = '0',
    MLIST,
    MBETA,
    MBIND,
    MBINR,
    MLETR,
    MCOND,
    MCONJ,
    MDISJ
};
```

The size of the node pool in nodes.

```
int Pool_size;
```

The arrays `Car`, `Cdr`, and `Tag` form the node pool. `Car` and `Cdr` hold the car and cdr fields of a cons cell, `Tag` holds the atom flag and the garbage collector tags. The car field of a node `n` is referred to by `Car[n]`, the cdr field of the same node by `Cdr[n]`, and its tags are accessed using `Tag[n]`.

Note that no pointers are used inside of the node pool. Each car and cdr field (except for atoms) contains the offset of another node in the array forming the node pool. Figure 9 depicts the internal representation of a cons cell. The cell is located at offset 5, which means that `Car[5]` holds its car field and `Cdr[5]` holds its cdr field. Its car field contains the integer 17, so its car value is stored in the node located at offset 17. Analogously, the value of the cdr part of the cons cell is stored in the node located at offset 29.

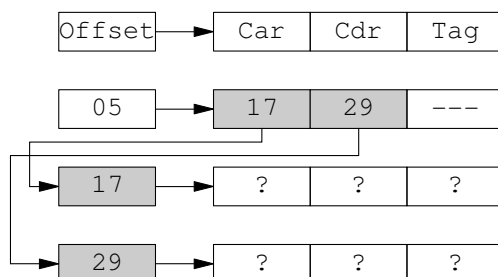


Fig. 9 – node pool structure

Because integer offsets are used for indirection instead of pointers, the entire node pool can be dumped to disk using basically three `write()` operations and restored at a later time using

`read()`. All addressing is done relative to the pool arrays.

```
int      *Car,                      /* Car*Cdr*Tag = Node Pool */
          *Cdr;
char      *Tag;
```

This is a cdr-linked list of free nodes.

```
int      Freelist;
```

The following variables are protected during garbage collections. A value that is bound to any of them will not be recycled. `Tmp_car` and `Tmp_cdr` are used to protect child nodes at allocation time. The other two are used whenever they are needed.

```
int      Tmp_car, Tmp_cdr;          /* GC-safe */
int      Tmp, Tmp2;
```

`Infile` is the name of the input file currently being read. A value of `NULL` means that terminal input is read by the REPL (read-eval-print loop). `Input` is the input stream itself.

`Rejected` is a character that has been put back to the input stream. When `Rejected=EOT`, no character has been rejected. Of course, `ungetc()` could have been used, but this solution is more transparent and easier ported to other languages.²²

`Line` is the current input line number relative to the beginning of the current input file. `Output` is the output stream to which all interpreter output (including error messages but excluding startup error messages) is sent.

```
char      *Infile;
FILE      *Input;
int      Rejected;
int      Line;
FILE      *Output;
```

The following string variables are used to buffer path names when **loading** programs. `Source_dir` is the directory from which source files are loaded, `Expanded_path` is used to expand files names beginning with a tilde, and `Current_path` captures the current path in nested **loads**. See the `load()` function for details.

```
char      Source_dir[MAX_PATH_LEN];
char      Expanded_path[MAX_PATH_LEN];
char      Current_path[MAX_PATH_LEN];
```

`Error_flag` and `Fatal_flag` are set when a (fatal) error occurs.

```
int      Error_flag;
struct Error_context
        Error;
```

²² The ancestor of `zenlisp` was written in T3X and not in C, which may explain some decisions that appear un-C-ish.

zen style programming

```
int      Fatal_flag;
```

`Symbols` is the global symbol table. `Safe_symbols` is a copy of the symbol table that is used to store a sane copy in case things go awfully wrong during reduction.

Because `zenlisp` uses shallow binding (which stores values directly in variables), the symbol table is merely a list of symbols.

```
int      Symbols;
int      Safe_symbols;
```

These are the various stacks that the interpreter uses when reducing expressions to their normal forms:

Stack general-purpose stack
Stack_bottom bottom of `Stack`, because `eval()` is reentrant
Mode_stack interpreter states
Arg_stack function arguments
Bind_stack bindings of **let** and **letrec**
Env_stack lexical environments of closures (performance hack)

```
int      Stack, Stack_bottom;
int      Mode_stack;
int      Arg_stack;
int      Bind_stack;
int      Env_stack;
```

These variables are used in error reporting and debugging:

```
int      Frame;
int      Function_name;
int      Traced_fn;
```

`Root` contains all variables whose values are to be protected during garbage collections. The collector will never recycle a value that is bound to any of these variables.

```
int      *Root[] = { &Symbols, &Stack, &Mode_stack, &Arg_stack, &Bind_stack,
                    &Env_stack, &Tmp_car, &Tmp_cdr, &Tmp, &Tmp2,
                    &Safe_symbols, NULL };
```

These variables are used to capture lexical environments. `Lexical_env` holds the lexical environment to build. `Bound_vars` contains the list of variables bound in a given context.

```
int      Lexical_env;
int      Bound_vars;
```

The next variables are used to keep track of the nesting levels of parentheses in the input as well as applications of **load** and `eval()`. The `eval()` function recurses internally during program reduction, but some functions (like **define**) are limited to the top level of recursion.

```
int    Paren_level;
int    Load_level;
int    Eval_level;
```

When `Quotedprint` is set to one, printed expressions are assumed to be quoted already, so no leading quote (apostrophy) will print. `Max_atoms_used` records the peak of the node usage during reduction. It is cleared by the **gc** function. `Max_trace` holds the maximal number of function names to print in the call trace in case of an error.

```
int    Quotedprint;
int    Max_atoms_used;
int    Max_trace;
```

When `Stat_flag` is set, reduction steps, nodes allocated, and garbage collections are counted. It is used internally by the **stats** pseudo function. `Closure_form` determines how much of a closure will print. 0 means only the arguments, 1 includes the body, 2 includes the lexical environment. `Verify_arrows` turns arrow verification on and off. `Verbose_GC` controls the output of GC statistics.

```
int    Stat_flag;
int    Closure_form;
int    Verify_arrows;
int    Verbose_GC;
```

These are counters for the **stats** pseudo function.

```
struct counter  Reductions,
                  Allocations,
                  Collections;
```

The following variables hold the offsets of frequently used symbols, so the symbols do not have to be looked up in the symbol table each time they are referred to. Some of the symbols are internal, some are `zenlisp` keywords.

```
int    S_bottom, S_closure, S_false, S_lambda, S_primitive,
        S_quote, S_special, S_special_cbv, S_true, S_void,
        S_last;
```

These are the opcodes of `zenlisp`'s primitive functions. They are offsets to the `Primitives` array, which holds pointers to the functions implementing the associated operations.

```
enum {  P_ATOM, P_BOTTOM, P_CAR, P_CDR, P_CONS, P_DEFINED, P_EQ,
        P_EXPLODE, P_GC, P_IMPLODE, P_QUIT, P_RECURSIVE_BIND,
        P_SYMBOLS, P_VERIFY_ARROWS, N_PRIMITIVES };
```

```
int    (*Primitives[N_PRIMITIVES])(int);
```

Pseudo function applications (special forms) are handled in the same way as primitive functions.

```
enum {  SF_AND, SF_APPLY, SF_CLOSURE_FORM, SF_COND, SF_DEFINE,
        SF_DUMP_IMAGE, SF_EVAL, SF_LAMBDA, SF_LET, SF_LETREC,
```


zen style programming

```
SF_LOAD, SF_OR, SF_QUOTE, SF_STATS, SF_TRACE,  
N_SPECIALS };
```

```
int      (*Specials[N_SPECIALS])(int, int *, int *, int *);
```

Stop lint from complaining about unused variables.

```
#ifndef LINT  
#define USE(arg)      (arg = NIL)  
#else  
#define USE(arg)  
#endif
```

Every single function in the program has a prototype. Feel free to skip ahead.

```
int      _rdch(void);  
int      add_primitive(char *name, int opcode);  
int      add_special(char *name, int opcode, int cbv);  
int      add_symbol(char *s, int v);  
int      alloc3(int pcar, int pcdr, int ptag);  
int      aunsave(int k);  
int      bad_argument_list(int n);  
void     bind_args(int n, int name);  
int      bunsave(int k);  
void     catch_int(int sig);  
void     clear_stats(void);  
void     collect_free_vars(int n);  
int      cond_get_pred(void);  
int      cond_eval_clause(int n);  
int      cond_setup(int n);  
int      copy_bindings(void);  
void     count(struct counter *c, int k);  
char     *counter_to_string(struct counter *c, char *buf);  
int      define_function(int n);  
int      dump_image(char *p);  
int      equals(int n, int m);  
void     eliminate_tail_calls(void);  
int      error(char *m, int n);  
int      eval(int n);  
char     *expand_path(char *s, char *buf);  
int      explode_string(char *sym);  
void     fatal(char *m);  
int      find_symbol(char *s);  
void     fix_all_closures(int b);  
void     fix_cached_closures(void);  
void     fix_closures_of(int n, int bindings);  
int      flat_copy(int n, int *lastp);  
int      gc(void);  
int      get_opt_val(int argc, char **argv, int *pi, int *pj, int *pk);  
void     get_options(int argc, char **argv);  
void     get_source_dir(char *path, char *pfx);  
char     *symbol_to_string(int n, char *b, int k);
```

202

```
void    help(void);
void    init(void);
void    init1(void);
void    init2(void);
int     is_alist(int n);
int     is_bound(int n);
int     is_list_of_symbols(int m);
void    let_bind(int env);
int     let_eval_arg(void);
int     let_finish(int rec);
int     let_next_binding(int n);
int     let_setup(int n);
int     load(char *p);
int     make_closure(int n);
void    mark(int n);
int     make_lexical_env(int term, int locals);
char    *make_zen_path(char *s);
int     munsave(void);
void    nl(void);
void    print(int n);
int     reverse_in_situ(int n);
void    pr(char *s);
int     primitive(int *np);
void    print_call_trace(int n);
int     print_closure(int n, int dot);
int     print_condensed_list(int n, int dot);
int     print_primitive(int n, int dot);
int     print_quoted_form(int n, int dot);
void    print_trace(int n);
void    print_license(void);
void    pr_num(int n);
int     quote(int n);
int     read_condensed(void);
void    read_eval_loop(void);
int     read_list(void);
int     read_symbol(int c);
void    repl(void);
void    reset_counter(struct counter *c);
void    reset_state(void);
void    restore_bindings(int values);
int     setup_and_or(int n);
int     special(int *np, int *pcf, int *pmode, int *pcbn);
int     string_to_symbol(char *s);
char    *symbol_to_string(int n, char *b, int k);
void    unbind_args(void);
int     unreadable(void);
int     unsave(int k);
void    usage(void);
void    verify(void);
int     wrong_arg_count(int n);
int     z_and(int n, int *pcf, int *pmode, int *pcbn);
```

zenstyle programming

```
int      z_apply(int n, int *pcf, int *pmode, int *pcbn);
int      z_atom(int n);
int      z_bottom(int n);
int      z_car(int n);
int      z_cdr(int n);
int      z_closure_form(int n, int *pcf, int *pmode, int *pcbn);
int      z_cond(int n, int *pcf, int *pmode, int *pcbn);
int      z_cons(int n);
int      z_define(int n, int *pcf, int *pmode, int *pcbn);
int      z_defined(int n);
int      z_dump_image(int n, int *pcf, int *pmode, int *pcbn);
int      z_eq(int n);
int      z_eval(int n, int *pcf, int *pmode, int *pcbn);
int      z_explode(int n);
int      z_gc(int n);
int      z_implode(int n);
int      z_lambda(int n, int *pcf, int *pmode, int *pcbn);
int      z_let(int n, int *pcf, int *pmode, int *pcbn);
int      z_letrec(int n, int *pcf, int *pmode, int *pcbn);
int      z_load(int n, int *pcf, int *pmode, int *pcbn);
int      z_or(int n, int *pcf, int *pmode, int *pcbn);
int      z_quit(int n);
int      z_quote(int n, int *pcf, int *pmode, int *pcbn);
int      z_recursive_bind(int n);
int      z_stats(int n, int *pcf, int *pmode, int *pcbn);
int      z_symbols(int n);
int      z_trace(int n, int *pcf, int *pmode, int *pcbn);
int      z_verify_arrows(int n);
int      zen_eval(int n);
void     zen_fini(void);
int      zen_init(int nodes, int trackGc);
char     **zen_license(void);
int      zen_load_image(char *p);
void     zen_print(int n);
void     zen_print_error(void);
int      zen_read(void);
void     zen_stop(void);
int      zread(void);
```

12.2 miscellaneous functions

These are convenience macros for accessing members of nested lists.

```
#define caar(x) (Car[Car[x]])
#define cadr(x) (Car[Cdr[x]])
#define cdar(x) (Cdr[Car[x]])
#define cddr(x) (Cdr[Cdr[x]])
#define caaar(x) (Car[Car[Car[x]]])
#define caadr(x) (Car[Car[Cdr[x]]])
#define cadar(x) (Car[Cdr[Car[x]])
#define caddr(x) (Car[Cdr[Cdr[x]])
```

```
#define cdaar(x) (Cdr[Car[Car[x]]])
#define cddar(x) (Cdr[Cdr[Car[x]]])
#define cdddr(x) (Cdr[Cdr[Cdr[x]]])
#define caddar(x) (Car[Cdr[Cdr[Car[x]]]])
#define cadddr(x) (Car[Cdr[Cdr[Cdr[x]]]])
```

All interpreter output (except for startup error messages) goes through this interface.

```
void nl(void) {
    putc('\n', Output);
    if (Output == stdout) fflush(Output);
}

void pr(char *s) {
    fputs(s, Output);
}

void pr_num(int n) {
    fprintf(Output, "%d", n);
}
```

12.3 error reporting

Print_call_trace prints a call trace if a non-empty call frame is passed to it.

```
void print_call_trace(int frame) {
    int s, n;

    s = frame;
    n = Max_trace;
    while (s != NIL) {
        if (n == 0 || Cdr[s] == NIL || cadr(s) == NIL) break;
        if (n == Max_trace) pr("* Trace:");
        n = n-1;
        pr(" ");
        Quotedprint = 1;
        print(cadr(s));
        s = Car[s];
    }
    if (n != Max_trace) nl();
}
```

Register an error context for later reporting and set the error flag. In case of multiple errors, register only the first one.

```
int error(char *m, int n) {
    if (Error_flag) return NIL;
    Error.msg = m;
    Error.expr = n;
    Error.file = Infile;
    Error.line = Line;
```

zen style programming

```
Error.fun = Function_name;
Error.frame = Frame;
Error_flag = 1;
return NIL;
}
```

Print the error message currently stored in the error context and clear the error flag.

```
void zen_print_error(void) {
    pr("* ");
    if (Error.file) {
        pr(Error.file);
        pr(": ");
    }
    pr_num(Error.line);
    pr(": ");
    if (Error.fun != NIL) {
        Quotedprint = 1;
        print(Error.fun);
    }
    else {
        pr("REPL");
    }
    pr(": ");
    pr(Error.msg);
    if (Error.expr != NO_EXPR) {
        if (Error.msg[0]) pr(": ");
        Quotedprint = 1;
        print(Error.expr);
    }
    nl();
    if (Error.arg) {
        pr("* ");
        pr(Error.arg); nl();
        Error.arg = NULL;
    }
    if (!Fatal_flag && Error.frame != NIL)
        print_call_trace(Error.frame);
    Error_flag = 0;
}
```

Report a fatal error and exit.

```
void fatal(char *m) {
    Error_flag = 0;
    Fatal_flag = 1;
    error(m, NO_EXPR);
    zen_print_error();
    pr("* Fatal error, aborting");
    nl();
    exit(1);
}
```

12.4 counting functions

```
void reset_counter(struct counter *c) {
    c->n = 0;
    c->n1k = 0;
    c->n1m = 0;
    c->n1g = 0;
}
```

Increment the counter **c** by **k**. Assert $0 \leq k \leq 1000$.

```
void count(struct counter *c, int k) {
    char *msg = "statistics counter overflow";

    c->n = c->n+k;
    if (c->n >= 1000) {
        c->n = c->n - 1000;
        c->n1k = c->n1k + 1;
        if (c->n1k >= 1000) {
            c->n1k = 0;
            c->n1m = c->n1m+1;
            if (c->n1m >= 1000) {
                c->n1m = 0;
                c->n1g = c->n1g+1;
                if (c->n1g >= 1000) {
                    error(msg, NO_EXPR);
                }
            }
        }
    }
}
```

Convert a counter structure to a string representation of the value stored in it. Commas will be inserted to mark thousands. A sufficiently large string buffer must be supplied by the caller. The greatest value that can be stored in a counter structure is 999,999,999,999.

```
char *counter_to_string(struct counter *c, char *buf) {
    int i;

    i = 0;
    if (c->n1g) {
        sprintf(&buf[i], "%d,", c->n1g);
        i = strlen(buf);
    }
    if (c->n1m || c->n1g) {
        if (c->n1g)
            sprintf(&buf[i], "%03d,", c->n1m);
        else
            sprintf(&buf[i], "%d,", c->n1m);
    }
}
```

zen style programming

```

        i = strlen(buf);
    }
    if (c->n1k || c->n1m || c->n1g) {
        if (c->n1g || c->n1m)
            sprintf(&buf[i], "%03d", c->n1k);
        else
            sprintf(&buf[i], "%d", c->n1k);
        i = strlen(buf);
    }
    if (c->n1g || c->n1m || c->n1k)
        sprintf(&buf[i], "%03d", c->n);
    else
        sprintf(&buf[i], "%d", c->n);
    return buf;
}

```

12.5 memory management

The `mark()` function implements a finite state machine (FSM) that traverses a tree rooted at the node `n`. The function marks all nodes that it encounters during traversal as “live” nodes, i.e. nodes that may not be recycled. The FSM uses three states (1,2,3) that are formed using the collector flags `MARK_FLAG` (M) and `SWAP_FLAG` (S). `MARK_FLAG` is a state flag and the “mark” flag — which is used to tag live nodes — at the same time. The following figures illustrate the states of the root node during the traversal of a tree of three nodes. Marked nodes are rendered with a grey background.

State 1: Node `N` is unvisited. The parent points to `NIL`, both flags are cleared.

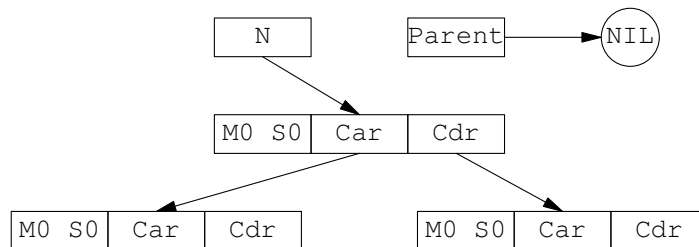


Fig. 10 – garbage collection, state 1

State 2: `N` now points to the car child of the root node, the parent pointer points to the root node, and the parent of the parent is stored in the car part of the root node. Both flags are set. The node is now marked.

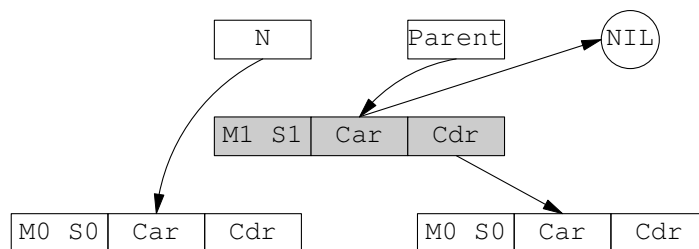


Fig. 11 – garbage collection, state 2

State 3: When the car child is completed, the car pointer of the root is restored, the grand-parent moves to the cdr part of the root node, and **N** moves to the cdr child. The **S** flag is cleared, and the root node is now completely traversed.

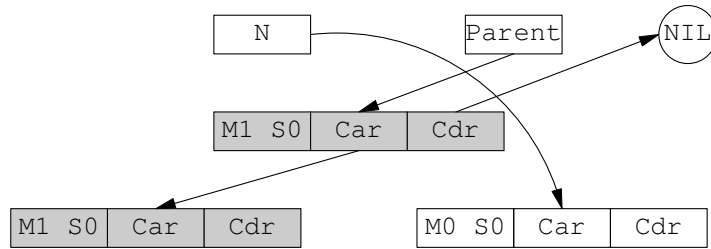


Fig. 12 – garbage collection, state 3

State 3: When the FSM returns from the cdr child, it finds the root node in state 3. To return to the root, it restores the cdr pointer of the root node and the parent. **N** moves up to the root node. Because **N** is marked and parent is NIL, the traversal is complete.

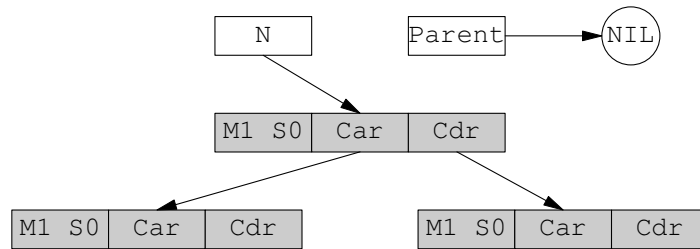


Fig. 13 – garbage collection, finished

When the FSM hits an already marked node during traversal, it returns to the parent node immediately. Because nodes get marked *before* their descendants are traversed, the FSM can traverse cyclic structures without entering an infinite loop.

When the mark phase finds an object with the **ATOM_FLAG** set, it traverses only its cdr field and leaves the car field alone (because it does not hold a valid node offset).

```
/*
 * Mark nodes which can be accessed through N.
 * Using the Deutsch/Schorr/Waite (aka pointer reversal) algorithm.
 * State 1: M==0 S==0 unvisited, process CAR
 * State 2: M==1 S==1 CAR visited, process CDR
 * State 3: M==1 S==0 completely visited, return to parent
 */
void mark(int n) {
    int    p, parent;

    parent = NIL;
    while (1) {
        if (n == NIL || Tag[n] & MARK_FLAG) {
            if (parent == NIL) break;
            if (Tag[parent] & SWAP_FLAG) { /* State 2 */
                /* Swap CAR and CDR pointers and */
                /* proceed with CDR. Go to State 3. */
                p = Cdr[parent];
                Cdr[parent] = Car[parent];
                Car[parent] = n;
            }
        }
    }
}
```


zen style programming

```

        Tag[parent] &= ~SWAP_FLAG;          /* S=0 */
        Tag[parent] |= MARK_FLAG;          /* M=1 */
        n = p;
    }
    else {                                  /* State 3: */
        /* Return to the parent and restore */
        /* parent of parent */
        p = parent;
        parent = Cdr[p];
        Cdr[p] = n;
        n = p;
    }
}
else {                                  /* State 1: */
    if (Tag[n] & ATOM_FLAG) {
        /* If this node is an atom, go directly */
        /* to State 3. */
        p = Cdr[n];
        Cdr[n] = parent;
        /*Tag[n] &= ~SWAP_FLAG;*/          /* S=0 */
        parent = n;
        n = p;
        Tag[parent] |= MARK_FLAG;          /* M=1 */
    }
    else {
        /* Go to state 2: */
        p = Car[n];
        Car[n] = parent;
        Tag[n] |= MARK_FLAG;                /* M=1 */
        parent = n;
        n = p;
        Tag[parent] |= SWAP_FLAG;           /* S=1 */
    }
}
}
}
}

```

Mark and Sweep garbage collector: first mark all `Root[]` nodes and the nodes of the error context (if necessary), then delete and rebuild the free list. Mark flags are cleared in the loop that builds the new free list.

```

int gc(void) {
    int    i, k;

    k = 0;
    for (i=0; Root[i]; i++) mark(Root[i][0]);
    if (Error_flag) {
        mark(Error.expr);
        mark(Error.fun);
        mark(Error.frame);
    }
}

```

```
Freelist = NIL;
for (i=0; i<Pool_size; i++) {
    if (!(Tag[i] & MARK_FLAG)) {
        Cdr[i] = Freelist;
        Freelist = i;
        k = k+1;
    }
    else {
        Tag[i] &= ~MARK_FLAG;
    }
}
if (Max_atoms_used < Pool_size-k) Max_atoms_used = Pool_size-k;
if (Verbose_GC) {
    pr_num(k);
    pr(" nodes reclaimed");
    nl();
}
if (Stat_flag) count(&Collections, 1);
return k;
}
```

The `alloc3()` function is the principal node allocator of `zenlisp`. It removes the first node of the free list and initializes it with the given car, cdr, and tag values. When the free list is empty, a garbage collection (GC) is triggered. Note that `alloc3()` protects the values passed to it from the GC, so no special care has to be taken by the caller. For example, the form `(x y z)` can be created using the code fragment:

```
    n = alloc(z, NIL);
    n = alloc(y, n);
    n = alloc(x, n);

int alloc3(int pcar, int pcdr, int ptag) {
    int    n;

    if (Stat_flag) count(&Allocations, 1);
    if (Freelist == NIL) {
        Tmp_cdr = pcdr;
        if (!ptag) Tmp_car = pcar;
        gc();
        Tmp_car = Tmp_cdr = NIL;
        if (Freelist == NIL) fatal("alloc3(): out of nodes");
    }
    n = Freelist;
    Freelist = Cdr[Freelist];
    Car[n] = pcar;
    Cdr[n] = pcdr;
    Tag[n] = ptag;
    return n;
}
```

`Alloc()` is a short cut for allocating cons cells.

zenstyle programming

```
#define alloc(pcar, pcdr) \  
    alloc3(pcar, pcdr, 0)
```

Save() saves a node on the stack, unsave() removes a given number of values and returns the deepest one removed by it.

```
#define save(n) \  
    (Stack = alloc(n, Stack))  
  
int unsave(int k) {  
    int    n;  
  
    USE(n);  
    while (k) {  
        if (Stack == NIL) fatal("unsave(): stack underflow");  
        n = Car[Stack];  
        Stack = Cdr[Stack];  
        k = k-1;  
    }  
    return n;  
}
```

Msave() and munsave() work like save() and unsave() above, but use the mode stack rather than the general purpose stack. Because Mode_stack holds integer values instead of nodes, the values are packaged in the car fields of atom nodes.

```
#define msave(v) \  
    (Car[Mode_stack] = alloc3(v, Car[Mode_stack], ATOM_FLAG))  
  
int munsave(void) {  
    int    v;  
  
    if (Car[Mode_stack] == NIL) fatal("munsave(): m-stack underflow");  
    v = caar(Mode_stack);  
    Car[Mode_stack] = cdar(Mode_stack);  
    return v;  
}
```

The following functions repeat the save/unsave procedure for the argument stack and binding stack, respectively.

```
#define asave(n) \  
    (Arg_stack = alloc(n, Arg_stack))  
  
int aunsave(int k) {  
    int    n;  
  
    USE(n);  
    while (k) {  
        if (Arg_stack == NIL) fatal("aunsave(): a-stack underflow");  
        n = Car[Arg_stack];  
    }
```

```
        Arg_stack = Cdr[Arg_stack];
        k = k-1;
    }
    return n;
}

#define bsave(n) \
    (Bind_stack = alloc(n, Bind_stack))

int bunsave(int k) {
    int    n;

    USE(n);
    while (k) {
        if (Bind_stack == NIL) fatal("bunsave(): b-stack underflow");
        n = Car[Bind_stack];
        Bind_stack = Cdr[Bind_stack];
        k = k-1;
    }
    return n;
}
```

12.6 symbol tables

Find a symbol with the name *s* in the global symbol table. Return the symbol or NIL if no such symbol exists.

```
int find_symbol(char *s) {
    int    p, n, i;

    p = Symbols;
    while (p != NIL) {
        n = caar(p);
        i = 0;
        while (n != NIL && s[i]) {
            if (s[i] != (Car[n] & 255)) break;
            n = Cdr[n];
            i = i+1;
        }
        if (n == NIL && !s[i]) return Car[p];
        p = Cdr[p];
    }
    return NIL;
}
```

Check whether a node is an atom in the sense of **atom**. *Note:* **atom** also classifies primitive functions, special form handlers and the **{void}** value as atoms. This is not covered by the `atomic()` function.

zen style programming

```
#define atomic(n) \
    ((n) == NIL || (Car[n] != NIL && (Tag[Car[n]] & ATOM_FLAG)))
```

The `symbolic()` function checks whether the given node represents a symbol. This is probably the right place to clarify what the difference between *atoms*, *symbols*, and *atomic nodes* is.

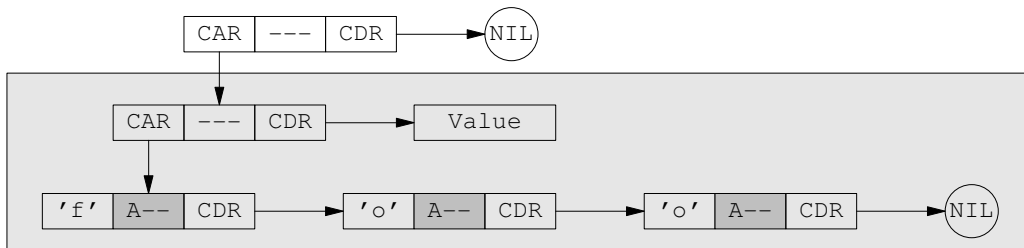


Fig. 14 – symbols, atoms, and atomic nodes

Figure 14 shows a list containing the symbol *foo* in so-called *box notation*. Each box containing three smaller boxes represents a node and each of the smaller boxes represents one field of that node. The first smaller box contains the car field, the second one the tag field, and the last one the cdr field. Three minus signs in a tag field indicate that no tags are set.

In Figure 14, the node outside of the large grey box is the “spine” of the form `(foo)`. Its cdr part points to `()` and its car part points to the symbol *foo*. Note that the symbol consists of (at least) four nodes: one that binds the symbol name to the value and three nodes that hold the characters of the symbol name.

A node with its `ATOM_FLAG` set is called an “atomic node”. It is used to hold some value, like a character of a symbol name, in its car field. A “symbol” is a node whose car part points to a chain of atomic nodes and whose cdr part points to a node tree that represents the value of that symbol. An “atom”, finally, is either a symbol or a primitive function, or `()`.

The `symbolic()` function checks whether a node is a symbol.

```
#define symbolic(n) \
    ((n) != NIL && Car[n] != NIL && (Tag[Car[n]] & ATOM_FLAG))
```

Create a symbol with the given name and return it.

```
int string_to_symbol(char *s) {
    int i, n, m, a;

    i = 0;
    if (s[i] == 0) return NIL;
    a = n = NIL;
    while (s[i]) {
        m = alloc3(s[i], NIL, ATOM_FLAG);
        if (n == NIL) {
            n = m;
        }
    }
}
```

```
                save(n);
            }
            else {
                Cdr[a] = m;
            }
            a = m;
            i = i+1;
        }
        unsave(1);
        return n;
    }
}
```

Create a string containing the name of the given symbol. When the symbol has a length of more than SYMBOL_LEN characters, an error is reported.

```
char *symbol_to_string(int n, char *b, int k) {
    int    i;

    n = Car[n];
    for (i=0; i<k-1; i++) {
        if (n == NIL) break;
        b[i] = Car[n];
        n = Cdr[n];
    }
    if (n != NIL) {
        error("symbol_to_string(): string too long", NO_EXPR);
        return NULL;
    }
    b[i] = 0;
    return b;
}
```

Add a symbol to the global symbol table. If a symbol with the given name already exists, return it without creating a new one.

```
int add_symbol(char *s, int v) {
    int    n, m;

    n = find_symbol(s);
    if (n != NIL) return n;
    n = string_to_symbol(s);
    m = alloc(n, v? v: n);
    Symbols = alloc(m, Symbols);
    return m;
}
```

Add a primitive function (add_primitive()) or special form handler (add_special()) to the current symbol table. Figure 15 outlines the internal structure of a primitive function.

zen style programming

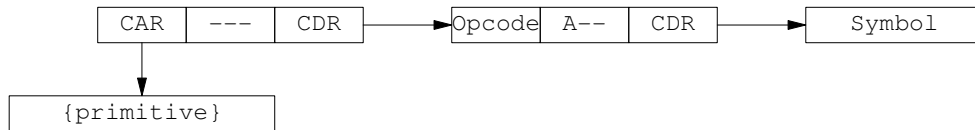


Fig. 15 - primitive function structure

The special symbol **{primitive}** marks the structure as a primitive function. The atom contains the opcode of the primitive and a link back to the name to which the primitive is bound. This allows the printer to output `{internal car}` rather than just `{internal}` when evaluating **car**.

Special form (SF) handlers have the same structure as primitive function handlers, but they use the **{special}** or **{special_cbv}** symbols instead of **{primitive}**. SF handlers using the **{special_cbv}** symbol are called by value. Other SF handlers are called by name.

```

int add_primitive(char *name, int opcode) {
    int y;

    y = add_symbol(name, 0);
    Cdr[y] = alloc(S_primitive, NIL);
    cddr(y) = alloc3(opcode, NIL, ATOM_FLAG);
    cdddr(y) = y;
    return y;
}

int add_special(char *name, int opcode, int cbv) {
    int y;

    y = add_symbol(name, 0);
    Cdr[y] = alloc(cbv? S_special_cbv: S_special, NIL);
    cddr(y) = alloc3(opcode, NIL, ATOM_FLAG);
    cdddr(y) = y;
    return y;
}

```

12.7 reader

All interpreter input goes through this interface.

```

int _rdch(void) {
    int c;

    if (Rejected != EOT) {
        c = Rejected;
        Rejected = EOT;
        return c;
    }
    c = getc(Input);
    if (feof(Input)) return EOT;
    if (c == '\n') Line = Line+1;
}

```

```
        return c;
    }

#define rdch() \
    tolower(_rdch())
```

The `read_list()` function reads proper lists and improper lists (including, of course, dotted pairs). It calls `zread()` to read each member of the list. `zread()` may call `read_list()` in turn to read sublists. When `read_list()` is invoked, the initial opening parenthesis already has been removed from the input.

`Read_read()` checks for properly formed improper lists (sic!) and reports all kinds of errors, like missing closing parentheses and dots in unexpected positions. It returns the list read in case of success and otherwise `()`.

```
int read_list(void) {
    int      n,
            lst,
            app,
            count;
    char     *badpair;

    badpair = "bad pair";
    Paren_level = Paren_level+1;
    lst = alloc(NIL, NIL); /* Root node */
    save(lst);
    app = NIL;
    count = 0;
    while (1) {
        if (Error_flag) {
            unsave(1);
            return NIL;
        }
        n = zread();
        if (n == EOT) {
            if (Load_level) return EOT;
            error("missing ' '", NO_EXPR);
        }
        if (n == DOT) {
            if (count < 1) {
                error(badpair, NO_EXPR);
                continue;
            }
            n = zread();
            Cdr[app] = n;
            if (n == R_PAREN || zread() != R_PAREN) {
                error(badpair, NO_EXPR);
                continue;
            }
            unsave(1);
        }
    }
}
```


zen style programming

```
        Paren_level = Paren_level-1;
        return lst;
    }
    if (n == R_PAREN) break;
    if (app == NIL)
        app = lst;
    else
        app = Cdr[app];
    Car[app] = n;
    Cdr[app] = alloc(NIL, NIL);
    count = count+1;
}
Paren_level = Paren_level-1;
if (app != NIL) Cdr[app] = NIL;
unsave(1);
return count? lst: NIL;
}
```

This function checks whether a given character is a *delimiter*. Delimiters separate individual tokens in the input stream.

```
#define is_delimiter(c) \
    ((c) == ' ' || \
     (c) == '\t' || \
     (c) == '\n' || \
     (c) == '\r' || \
     (c) == '(' || \
     (c) == ')' || \
     (c) == ';' || \
     (c) == '.' || \
     (c) == '#' || \
     (c) == '{' || \
     (c) == '\')
```

Read a condensed list (excluding the introducing “#” character) and return it.

```
int read_condensed(void) {
    int    n, c, a;
    char   s[2];

    n = alloc(NIL, NIL);
    save(n);
    a = NIL;
    s[1] = 0;
    c = rdch();
    while (!is_delimiter(c)) {
        if (a == NIL) {
            a = n;
        }
        else {
            Cdr[a] = alloc(NIL, NIL);

```

```
        a = Cdr[a];
    }
    s[0] = c;
    Car[a] = add_symbol(s, S_void);
    c = rdch();
}
unsave(1);
Rejected = c;
return n;
}
```

Convert a string to a list of single-character symbols (a condensed list). Return the resulting list. This function is used internally to convert numeric values to a form that can be printed by the `zenlisp` printer, e.g. "12345" → #12345.

```
int explode_string(char *sym) {
    int    n, a, i;
    char    s[2];

    n = alloc(NIL, NIL);
    save(n);
    a = NIL;
    s[1] = 0;
    i = 0;
    while (sym[i]) {
        if (a == NIL) {
            a = n;
        }
        else {
            Cdr[a] = alloc(NIL, NIL);
            a = Cdr[a];
        }
        s[0] = sym[i];
        Car[a] = add_symbol(s, S_void);
        i += 1;
    }
    unsave(1);
    return n;
}
```

Quote the node **n**.

```
int quote(int n) {
    int    q;

    q = alloc(n, NIL);
    return alloc(S_quote, q);
}
```

Read a symbol and return it. When the symbol is not yet the symbol table, add it.

zenstyle programming

```
int read_symbol(int c) {
    char    s[SYMBOL_LEN];
    int     i;

    i = 0;
    while (!is_delimiter(c)) {
        if (i >= SYMBOL_LEN-2) {
            error("symbol too long", NO_EXPR);
            i = i-1;
        }
        s[i] = c;
        i = i+1;
        c = rdch();
    }
    s[i] = 0;
    Rejected = c;
    return add_symbol(s, S_void);
}
```

Check whether two forms are equal. This function is equal to the **equal** function of zenlisp. Because **equal** is written in zenlisp, it is not yet available at this point.

```
int equals(int n, int m) {
    if (n == m) return 1;
    if (n == NIL || m == NIL) return 0;
    if (Tag[n] & ATOM_FLAG || Tag[m] & ATOM_FLAG) return 0;
    return equals(Car[n], Car[m])
        && equals(Cdr[n], Cdr[m]);
}
```

The `verify()` function reads a form that follows a `=>` operator and compares it to the normal form of the most recently reduced expression. When the forms differ, it reports an error.

```
void verify(void) {
    int     expected;

    expected = zread();
    if (!atomic(expected) && Car[expected] == S_quote)
        expected = cadr(expected);
    if (!equals(expected, Cdr[S_last]))
        error("Verification failed; expected", expected);
}
```

Read an unreadable object (sic!) and report it.

```
int unreadable(void) {
    #define L 256
    int     c, i;
    static char    b[L];

    i = 0;
    b[0] = '{';
```

```
    c = '{';
    while (c != '}' && c != EOT && i < L-2) {
        b[i++] = c;
        c = rdch();
    }
    b[i] = '>';
    b[i+1] = 0;
    Error.arg = b;
    return error("unreadable object", NO_EXPR);
}
```

Zread() is the **zenlisp** reader interface. It reads a form from the input stream and returns a tree of nodes that is the internal representation of that form. It also skips over comments and evaluates => operators. When the end of the input stream has been reached, calling zread() yields EOT.

```
int zread(void) {
    int c;

    c = rdch();
    while (1) {
        while (c == ' ' || c == '\t' || c == '\n' || c == '\r') {
            if (Error_flag) return NIL;
            c = rdch();
        }
        if (c == '=' && Paren_level == 0) {
            c = rdch();
            if (c != '>') {
                Rejected = c;
                c = '=';
                break;
            }
            if (Verify_arrows) verify();
        }
        else if (c != ';') {
            break;
        }
        while (c != '\n') c = rdch();
    }
    if (c == EOT) return EOT;
    if (c == '(') {
        return read_list();
    }
    else if (c == '\\') {
        return quote(zread());
    }
    else if (c == '#') {
        return read_condensed();
    }
    else if (c == ')') {
        if (!Paren_level) return error("unexpected ')'", NO_EXPR);
        return R_PAREN;
    }
}
```

zen style programming

```
    }
    else if (c == '.') {
        if (!Paren_level) return error("unexpected '.'", NO_EXPR);
        return DOT;
    }
    else if (c == '{') {
        return unreadable();
    }
    else {
        return read_symbol(c);
    }
}
```

12.8 primitive operation handlers

A *primitive function* of zenlisp is a LISP function that is implemented in C. Its implementation is called a “*primitive operation handler*”.

This section discusses the primitive functions of zenlisp. Each primitive operation handler receives an expression and returns its value. The arguments of primitives already are in their normal forms at this point.

These functions are short cuts for reporting common errors.

```
int wrong_arg_count(int n) {
    return error("wrong argument count", n);
}

int bad_argument_list(int n) {
    return error("bad argument list", n);
}
```

Zenlisp prototypes like the **cons** prototype below have the following general form:

(function argument₁ ...) → result₁ | result₂

Function is the described function, each **argument** specifies the type of one argument, **result** is the type of the resulting normal form, and ... indicates that zero or more instances of the preceding datum may occur. The arrow denotes a mapping from the argument types to the type of the normal form. The vertical bar (“|”) represents a logical “or”. It may occur in argument lists, too.

Note that prototypes are not part of the zenlisp language. They are only used to describe functions in a formal way.

(cons form form) → pair

```
int z_cons(int n) {
    int    m, m2;
```

```
    m = Cdr[n];
    if (m == NIL || Cdr[m] == NIL || caddr(m) != NIL)
        return wrong_arg_count(n);
    m2 = cadr(m);
    m = alloc(Car[m], m2);
    return m;
}
```

(car pair) → form

```
int z_car(int n) {
    int m;

    m = Cdr[n];
    if (m == NIL || Cdr[m] != NIL) return wrong_arg_count(n);
    m = Car[m];
    if (    atomic(m) ||
          Car[m] == S_primitive ||
          Car[m] == S_special ||
          Car[m] == S_special_cbv
        )
        return error("car: cannot split atoms", m);
    return Car[m];
}
```

(cdr pair) → form

```
int z_cdr(int n) {
    int m;

    m = Cdr[n];
    if (m == NIL || Cdr[m] != NIL) return wrong_arg_count(n);
    m = Car[m];
    if (    atomic(m) ||
          Car[m] == S_primitive ||
          Car[m] == S_special ||
          Car[m] == S_special_cbv
        )
        return error("cdr: cannot split atoms", m);
    return Cdr[m];
}
```

(eq form₁ form₂) → :t | :f

```
int z_eq(int n) {
    int m;

    m = Cdr[n];
    if (m == NIL || Cdr[m] == NIL || caddr(m) != NIL)
        return wrong_arg_count(n);
    return Car[m] == cadr(m)? S_true: S_false;
}
```

zenstyle programming

(atom form) → :t | :f

Note that **atom** also returns **:t** for primitive functions, special form handlers and **{void}**.

```
int z_atom(int n) {
    int    m;

    m = Cdr[n];
    if (m == NIL || Cdr[m] != NIL) return wrong_arg_count(n);
    if atomic(Car[m]) return S_true;
    m = caar(m);
    return (m == S_primitive || m == S_special ||
           m == S_special_cbv || m == S_void)? S_true: S_false;
}
```

(explode symbol) → list

```
int z_explode(int n) {
    int    m, y, a;
    char    s[2];

    m = Cdr[n];
    if (m == NIL || Cdr[m] != NIL) return wrong_arg_count(n);
    m = Car[m];
    if (m == NIL) return NIL;
    if (!symbolic(m)) return error("explode: got non-symbol", m);
    y = alloc(NIL, NIL);
    save(y);
    a = y;
    m = Car[m];
    s[1] = 0;
    while (m != NIL) {
        s[0] = Car[m];
        Car[a] = add_symbol(s, S_void);
        m = Cdr[m];
        if (m != NIL) {
            Cdr[a] = alloc(NIL, NIL);
            a = Cdr[a];
        }
    }
    unsave(1);
    return y;
}
```

(implode list) → symbol

```
int z_implode(int n) {
    int    m, i;
    char    s[SYMBOL_LEN];

    m = Cdr[n];
    if (m == NIL || Cdr[m] != NIL) return wrong_arg_count(n);
```

```
m = Car[m];
if (m == NIL) return NIL;
i = 0;
while (m != NIL) {
    if (!symbolic(Car[m]))
        return error("implode: non-symbol in argument",
                      Car[m]);
    if (cdaar(m) != NIL)
        return error(
            "implode: input symbol has multiple characters",
            Car[m]);
    if (i >= SYMBOL_LEN-1)
        return error("implode: output symbol too long", m);
    s[i] = caaar(m);
    i += 1;
    m = Cdr[m];
}
s[i] = 0;
return add_symbol(s, S_void);
}
```

The following functions deal with recursive lexical environments. They are used by the **recursive-bind** function and the **letrec** special form.

The `fix_cached_closures()` function fixes recursive bindings created by **letrec**. When a recursive function is created using **letrec**, **lambda** closes over the function name before binding the function to its name. This is demonstrated here using **let** instead of **letrec**:

```
(closure-form env)
(let ((f (lambda (x) (f x)))) f) => (closure #x #fx ((f . {void})))
```

Whenever a closure is created by **lambda**, the lexical environment of that closure is saved on the environment stack (`Env_stack`). When **letrec** finishes creating its bindings, it calls `fix_cached_closures()` in order to fix recursive bindings of the above form. To delimit its scope **letrec** pushes `:t` to the `Env_stack` before starting to process bindings.

When `fix_cached_closures()` is called, the `Env_stack` contains a list of lexical environments like `((f . {void}))` above. The car of the binding stack (`Bind_stack`) contains a list of symbols bound by **letrec**.

What `fix_cached_closures()` does now is to traverse each lexical environment in the topmost scope of `Env_stack` and check whether any of its variables is contained in the list on `Bind_stack`. When it finds such a binding, it changes the value associated with that variable in the environment with the value of the symbol on `Bind_stack`. Thereby it changes the above example as follows:

```
(closure #x #fx ((f . {void}))) -> (closure #x #fx ((f . fouter)))
```


zen style programming

Here f_{outer} refers to the outer binding of f (from `Bind_stack`). Because the outer f binds to the structure containing the fixed environment, a recursive structure is created.

```
void fix_cached_closures(void) {
    int    a, ee, e;

    if (Error_flag || Env_stack == NIL || Env_stack == S_true) return;
    a = Car[Bind_stack];
    while (a != NIL) {
        ee = Env_stack;
        while (ee != NIL && ee != S_true) {
            e = Car[ee];
            while (e != NIL) {
                if (Car[a] == caar(e)) {
                    cdar(e) = cdar(a);
                    break;
                }
                e = Cdr[e];
            }
            ee = Cdr[ee];
        }
        a = Cdr[a];
    }
}
```

Check whether **n** is an association list (an “alist”).

```
int is_alist(int n) {
    if (symbolic(n)) return 0;
    while (n != NIL) {
        if (symbolic(Car[n]) || !symbolic(caar(n)))
            return 0;
        n = Cdr[n];
    }
    return 1;
}
```

The following function is like `fix_cached_closures()`, but instead of fixing cached environments (on the `Env_stack`), it traverses all bindings of a given environment **n** and fixes the bindings of the symbols explicitly specified in **bindings**.

```
void fix_closures_of(int n, int bindings) {
    int    ee, e;
    int    bb, b;

    if (atomic(n)) return;
    if (Car[n] == S_closure) {
        fix_closures_of(caddr(n), bindings);
        ee = cdddr(n);
        if (ee == NIL) return;
        ee = Car[ee];
    }
}
```

```
        while (ee != NIL) {
            e = Car[ee];
            bb = bindings;
            while (bb != NIL) {
                b = Car[bb];
                if (Car[b] == Car[e])
                    Cdr[e] = Cdr[b];
                bb = Cdr[bb];
            }
            ee = Cdr[ee];
        }
        return;
    }
    fix_closures_of(Car[n], bindings);
    fix_closures_of(Cdr[n], bindings);
}
```

Fix all closures of a given environment.

```
void fix_all_closures(int b) {
    int    p;

    p = b;
    while (p != NIL) {
        fix_closures_of(cdar(p), b);
        p = Cdr[p];
    }
}
```

(recursive-bind alist) → alist

Side effect: create cyclic bindings.

```
int z_recursive_bind(int n) {
    int    m, env;

    m = Cdr[n];
    if (m == NIL || Cdr[m] != NIL) return wrong_arg_count(n);
    env = Car[m];
    if (!is_alist(env))
        return error("recursive-bind: bad environment", env);
    fix_all_closures(env);
    return env;
}
```

(bottom ...) → undefined

Side effect: stop reduction and report an error.

```
int z_bottom(int n) {
    n = alloc(S_bottom, Cdr[n]);
    return error("", n);
}
```

zen style programming

(defined symbol) → :t | :f

```
int z_defined(int n) {
    int    m;

    m = Cdr[n];
    if (m == NIL || Cdr[m] != NIL) return wrong_arg_count(n);
    if (!symbolic(Car[m]))
        return error("defined: got non-symbol", Car[m]);
    return cdar(m) == S_void? S_false: S_true;
}
```

(gc) → ' (free-nodes peak-usage)

```
int z_gc(int n) {
    int    m;
    char    s[20];

    m = Cdr[n];
    if (m != NIL) return wrong_arg_count(n);
    n = alloc(NIL, NIL);
    save(n);
    sprintf(s, "%d", gc());
    Car[n] = explode_string(s);
    Cdr[n] = alloc(NIL, NIL);
    sprintf(s, "%d", Max_atoms_used);
    Max_atoms_used = 0;
    cadr(n) = explode_string(s);
    unsave(1);
    return n;
}
```

(quit) → undefined

Side effect: exit from interpreter.

```
int z_quit(int n) {
    int    m;

    m = Cdr[n];
    if (m != NIL) return wrong_arg_count(n);
    zen_fini();
    exit(0);
}
```

(symbols) → ' (symbol ...)

```
int z_symbols(int n) {
    int    m;

    m = Cdr[n];
    if (m != NIL) return wrong_arg_count(n);
    return Symbols;
}
```

(verify-arrows :t | :f) → :t | :f

Side effect: turn arrow verification on or off.

```
int z_verify_arrows(int n) {
    int      m;

    m = Cdr[n];
    if (m == NIL || Cdr[m] != NIL) return wrong_arg_count(n);
    m = Car[m];
    if (m != S_true && m != S_false)
        return error("verify-arrows: got non truth-value", m);
    Verify_arrows = m == S_true;
    return m;
}
```

If `Car[np[0]]` is a primitive function, run the corresponding primitive operation handler, set `np[0]` to the result of the operation, and return 1. If `Car[np[0]]` is not a primitive function, return 0.

```
int primitive(int *np) {
    int      n, y;
    int      (*op)(int);

    n = np[0];
    y = Car[n];
    if (Error_flag) return 0;
    if (Car[y] == S_primitive) {
        op = Primitives[cadr(y)];
    }
    else {
        return 0;
    }
    n = (*op)(n);
    np[0] = n;
    return 1;
}
```

12.9 special form handlers

A *special form handler* is a function that handles the interpretation of a “special form”. *Special forms* are those forms that constitute the syntax of `zenlisp`. They are applications of keywords like **lambda**, **define**, and **cond**.

Each special form handler receives four arguments: the special form **n** and three pointers to **int** variables named **pcf**, **pmode**, and **pcbn**. These variables form the *state of the evaluator*.

The handler rewrites the form **n** in a way that is specific to the special form and returns it. The pointers **pcf**, **pmode**, and **pcbn** are used to control what the core of the evaluator does with the rewritten form. **Pmode** is the new mode of the evaluator, **pcbn** controls whether the returned form

zen style programming

is evaluated using call-by-name, and **pcf** is the so-called *continue flag*. Setting `pcf[0]` signals the evaluator that the returned form is an expression rather than a value. In this case the evaluation of the form must continue. Hence the name of this flag.

Special form handlers are also responsible for checking the syntax of the forms passed to them.

The `setup_and_or()` function prepares an **and** or **or** form for reduction.

Things get a bit messy at this point, because the `Bind_stack` is used to store temporary results of control constructs, too. In the following case, it keeps the argument list of **and** or **or**.

```
int setup_and_or(int n) {
    int m;

    m = Cdr[n];
    if (m == NIL) return wrong_arg_count(n);
    bsave(m);
    return Car[m];
}
```

(and expression ...) → form

```
int z_and(int n, int *pcf, int *pmode, int *pcbn) {
    USE(pcbn);
    if (Cdr[n] == NIL) {
        return S_true;
    }
    else if (cddr(n) == NIL) {
        *pcf = 1;
        return cadr(n);
    }
    else {
        *pcf = 2;
        *pmode = MCONJ;
        return setup_and_or(n);
    }
}
```

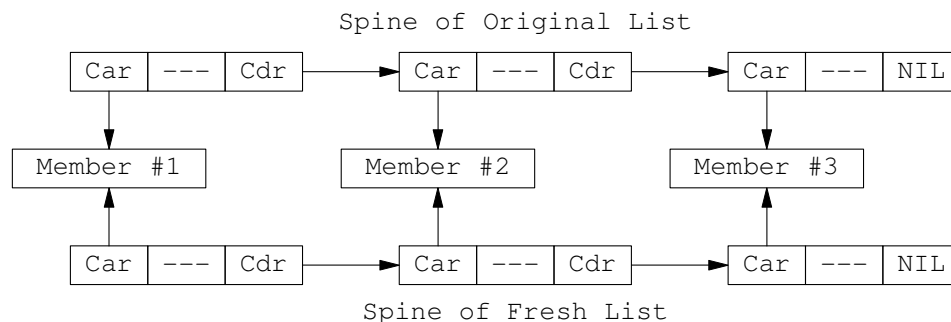


Fig. 16 – shared list

The `flat_copy()` function creates a fresh list that consists of the same objects as the list that was passed to it. Only the nodes that form the *spine* of the list are allocated freshly. Both the original list and the fresh list share the same members. Figure 16 illustrates this principle.

Note that instead of appending a pointer to `()` in above diagram, `NIL` is included in the `cdr` part of the last box of each list for brevity.

```
int flat_copy(int n, int *lastp) {
    int    a, m, last;

    if (n == NIL) {
        lastp[0] = NIL;
        return NIL;
    }
    m = alloc(NIL, NIL);
    save(m);
    a = m;
    last = m;
    while (n != NIL) {
        Car[a] = Car[n];
        last = a;
        n = Cdr[n];
        if (n != NIL) {
            Cdr[a] = alloc(NIL, NIL);
            a = Cdr[a];
        }
    }
    unsave(1);
    lastp[0] = last;
    return m;
}
```

(apply function [expression ...] list) → form

This handler merely rewrites

(apply function [expression ...] list)

to

(function [expression ...] . list)

and returns it. Note the dot before the *list* argument! When the **apply** handler finishes, the evaluator re-reduces the returned expression.

```
int z_apply(int n, int *pcf, int *pmode, int *pcbn) {
    int    m, p, q, last;
    char    *err1 = "apply: got non-function",
            *err2 = "apply: improper argument list";

    *pcf = 1;
    USE(pmode);
    *pcbn = 1;
```

zen style programming

```
m = Cdr[n];
if (m == NIL || Cdr[m] == NIL) return wrong_arg_count(n);
if (atomic(Car[m])) return error(err1, Car[m]);
p = caar(m);
if (    p != S_primitive &&
        p != S_special &&
        p != S_special_cbv &&
        p != S_closure
    )
    return error(err1, Car[m]);
p = Cdr[m];
USE(last);
while (p != NIL) {
    if (symbolic(p)) return error(err2, cadr(m));
    last = p;
    p = Cdr[p];
}
p = Car[last];
while (p != NIL) {
    if (symbolic(p)) return error(err2, Car[last]);
    p = Cdr[p];
}
if (caddr(m) == NIL) {
    p = cadr(m);
}
else {
    p = flat_copy(Cdr[m], &q);
    q = p;
    while (caddr(q) != NIL) q = Cdr[q];
    Cdr[q] = Car[last];
}
return alloc(Car[m], p);
}
```

Extract the predicate of the current clause of a **cond** expression. Car[Bind_stack] holds the clauses.

```
int cond_get_pred(void) {
    int    e;

    e = caar(Bind_stack);
    if (atomic(e) || atomic(Cdr[e]) || caddr(e) != NIL)
        return error("cond: bad clause", e);
    return Car[e];
}
```

Prepare a **cond** expression for evaluation. Save the clauses on the Bind_stack and return the first predicate.

```
int cond_setup(int n) {
    int    m;
```

```

    m = Cdr[n];
    if (m == NIL) return wrong_arg_count(n);
    bsave(m);
    return cond_get_pred();
}

```

Evaluate next clause of **cond**. **N** is the value of the current predicate. If **n = f**, return the predicate of the next clause. If **n ≠ f**, return the expression associated with that predicate (the body of the clause). When returning the body of a clause, set the context on **Bind_stack** to **()** to signal that the evaluation of the **cond** expression is complete.

```

int cond_eval_clause(int n) {
    int e;

    e = Car[Bind_stack];
    if (n == S_false) {
        Car[Bind_stack] = Cdr[e];
        if (Car[Bind_stack] == NIL)
            return error("cond: no default", NO_EXPR);
        return cond_get_pred();
    }
    else {
        e = cadar(e);
        Car[Bind_stack] = NIL;
        return e;
    }
}

```

(cond (predicate₁ expression₁)
(predicate₂ expression₂)
...) → form

```

int z_cond(int n, int *pcf, int *pmode, int *pcbn) {
    *pcf = 2;
    *pmode = MCOND;
    USE(pcbn);
    return cond_setup(n);
}

```

Check whether **m** is a list of symbols.

```

int is_list_of_symbols(int m) {
    while (m != NIL) {
        if (!symbolic(Car[m])) return 0;
        if (symbolic(Cdr[m])) break;
        m = Cdr[m];
    }
    return 1;
}

```


zen style programming

(define (symbol₁ symbol₂ ...) expression) → symbol

This function rewrites function definitions of the above form to

(lambda (symbol₂ ...) expression)

evaluates that expression and binds its normal form to **symbol₁**.

Side effect: create global binding.

```
int define_function(int n) {
    int      m, y;

    m = Cdr[n];
    if (Car[m] == NIL)
        return error("define: missing function name",
                      Car[m]);
    if (!is_list_of_symbols(Car[m])) return bad_argument_list(Car[m]);
    y = caar(m);
    save(cadr(m));
    Tmp2 = alloc(S_lambda, NIL);
    Cdr[Tmp2] = alloc(cdar(m), NIL);
    cddr(Tmp2) = alloc(cadr(m), NIL);
    cdddr(Tmp2) = alloc(NIL, NIL);
    Cdr[y] = eval(Tmp2);
    Tmp2 = NIL;
    unsave(1);
    return y;
}
```

(define (symbol₁ symbol₂ ...) expression) → symbol

(define symbol expression) → symbol

Evaluate an expression and bind its normal form to a symbol. If the expression is a **lambda** special form, create a closure with an *empty* lexical environment, thereby effectively implementing dynamic scoping.

Side effect: create global binding.

```
int z_define(int n, int *pcf, int *pmode, int *pcbn) {
    int      m, v, y;

    USE(pcf);
    USE(pmode);
    USE(pcbn);
    if (Eval_level > 1) {
        error("define: limited to top level", NO_EXPR);
        return NIL;
    }
    m = Cdr[n];
    if (m == NIL || Cdr[m] == NIL || cddr(m) != NIL)
        return wrong_arg_count(n);
    y = Car[m];
    if (!symbolic(y)) return define_function(n);
    v = cadr(m);
```

```
save(v);
/* If we are binding to a lambda expression, */
/* add a null environment */
if (!atomic(v) && Car[v] == S_lambda) {
    if (Cdr[v] != NIL && caddr(v) != NIL &&
        caddr(v) == NIL
    ) {
        caddr(v) = alloc(NIL, NIL);
    }
}
Cdr[y] = eval(cadr(m));
unsave(1);
return y;
}
```

(eval expression) → form

The `z_eval()` function just returns the expression passed to it for further reduction. Hence `((lambda (x) (x x)) (lambda (x) (eval '(x x))))` reduces in constant space.

```
int z_eval(int n, int *pcf, int *pmode, int *pcbn) {
    int m;

    *pcf = 1;
    USE(pmode);
    *pcbn = 0;
    m = Cdr[n];
    if (m == NIL || Cdr[m] != NIL) return wrong_arg_count(n);
    return (Car[m]);
}
```

The following functions deal with closure generation. They are used by the **lambda** special form.

The `is_bound()` function checks whether the symbol `n` is bound in the current context. The context is specified by the variables `Bound_vars`, which contains a (potentially improper) list of variables, and `Car[Lexical_env]`, which holds the lexical environment built so far. `Is_bound()` returns a flag indicating whether the variable is bound.

```
int is_bound(int n) {
    int b;

    b = Bound_vars;
    while (b != NIL) {
        if (symbolic(b)) {
            if (n == b) return 1;
            break;
        }
        if (n == Car[b]) return 1;
        b = Cdr[b];
    }
}
```

zen style programming

```
b = Car[Lexical_env];
while (b != NIL) {
    if (caar(b) == n) return 1;
    b = Cdr[b];
}
return 0;
}
```

`collect_free_vars()` collects the free variables of a lambda expression. This function expects an empty environment in the `car` field of `Lexical_env` and a list of bound variables in `Bound_vars`. It returns nothing, but builds the lexical environment in `Car[Lexical_env]`.

The function does not traverse expressions that begin with the keyword **quote**. To do so, it is not sufficient to just check for `Car[n] == S_quote`, because doing so would also catch expressions like **(list quote foo)**. By checking `caar(n)` instead, it makes sure that **quote** actually is in a `car` position. **Note:** this also prevents `(quote . {internal quote})` from being included, but who wants to re-define **quote** anyway?

```
void collect_free_vars(int n) {
    if (n == NIL || (Tag[n] & ATOM_FLAG)) return;
    if (symbolic(n)) {
        if (is_bound(n)) return;
        Car[Lexical_env] = alloc(NIL, Car[Lexical_env]);
        caar(Lexical_env) = alloc(n, Car[n] == Cdr[n]? n: Cdr[n]);
        return;
    }
    if (atomic(Car[n]) || caar(n) != S_quote)
        collect_free_vars(Car[n]);
    collect_free_vars(Cdr[n]);
}
```

The following function creates a lexical environment (a.k.a *lexical context*) for a closure. It is not sufficient to capture the current environment, because the interpreter uses *shallow binding*, where values are stored directly in symbols. `Make_lexical_env()` traverses the function term `term` and collects all free variables contained in it. `Locals` is the argument list of the lambda form to convert to a closure. Local variables are not collected because they are bound by **lambda**.

```
int make_lexical_env(int term, int locals) {
    Lexical_env = alloc(NIL, NIL);
    save(Lexical_env);
    Bound_vars = locals;
    collect_free_vars(term);
    unsave(1);
    return Car[Lexical_env];
}
```

Convert a lambda form to a closure:

```
(lambda (symbol ...) expression)
→ (closure (symbol ...) expression alist)
```

Save the environment of the closure on the Env_stack so it can be fixed by fix_cached_closures().

```
int make_closure(int n) {
    int    cl, env, args, term;

    if (Error_flag) return NIL;
    args = cadr(n);
    term = caddr(n);
    if (cddddr(n) == NIL) {
        env = make_lexical_env(term, args);
        if (env != NIL) {
            if (Env_stack != NIL)
                Env_stack = alloc(env, Env_stack);
            cl = alloc(env, NIL);
        }
        else {
            cl = NIL;
        }
    }
    else {
        cl = alloc(caddr(n), NIL);
    }
    cl = alloc(term, cl);
    cl = alloc(args, cl);
    cl = alloc(S_closure, cl);
    return cl;
}
```

(lambda (symbol ...) expression) → {closure ...}

```
int z_lambda(int n, int *pcf, int *pmode, int *pcbn) {
    int    m;

    m = Cdr[n];
    if (    m == NIL || Cdr[m] == NIL ||
        (caddr(m) != NIL && cddddr(m) != NIL)
    )
        return wrong_arg_count(n);
    if (caddr(m) != NIL && !is_alist(caddr(m)))
        return error("lambda: bad environment",
            caddr(m));
    if (!symbolic(Car[m]) && !is_list_of_symbols(Car[m]))
        return bad_argument_list(Car[m]);
    return Car[n] == S_closure? n: make_closure(n);
}
```

The unbind_args() function is used by the evaluator to restore the call frame of the caller of the current function. It restores the frame pointer (Frame), the name of the currently active function (Function_name), and the outer values of all symbols bound in the current frame.

zen style programming

```
void unbind_args(void) {
    int    v;

    Frame = unsave(1);
    Function_name = unsave(1);
    v = bunsave(1);
    while (v != NIL) {
        cdar(v) = unsave(1);
        v = Cdr[v];
    }
}
```

The next function sets up a context for the reduction of **let** and **letrec** special forms. The function saves the complete special form, an additional copy of the environment, an (initially empty) list of symbols to re-bind, and an empty set of inner bindings on Bind_stack. It also saves Env_stack on Stack and creates an empty Env_stack. It returns the environment to be processed (the first argument of the special form).

```
int let_setup(int n) {
    int    m;

    m = Cdr[n];
    if (m == NIL || Cdr[m] == NIL || caddr(m) != NIL)
        return wrong_arg_count(n);
    m = Car[m];
    if (symbolic(m))
        return error("let/letrec: bad environment", m);
    bsave(n);          /* save entire LET/LETREC */
    bsave(m);          /* save environment */
    bsave(NIL);        /* list of bindings */
    bsave(NIL);        /* save empty name list */
    save(Env_stack); /* get outer bindings out of the way */
    Env_stack = NIL;
    return m;
}
```

Process one binding of **let/letrec**. Add the current binding to the list of new bindings and remove it from the environment of Bind_stack. Return the rest of the environment. When this function returns **()**, all bindings have been processed.

```
int let_next_binding(int n) {
    int    m, p;

    m = caddr(Bind_stack); /* rest of environment */
    if (m == NIL) return NIL;
    p = Car[m];
    Tmp2 = n;
    cadr(Bind_stack) = alloc(NIL, cadr(Bind_stack));
    caadr(Bind_stack) = alloc(Car[p], n);
    Tmp2 = NIL;
}
```

```
        caddr(Bind_stack) = Cdr[m];  
        return Cdr[m];  
    }
```

Evaluate one argument of **let/letrec**. Fetch one binding from the environment saved on `Bind_stack` and check its syntax. If the syntax is wrong, clean up the context and bail out. If the binding is well-formed, save its variable (car field) in the name list on `Bind_stack` and return the associated expression (cadr field) for reduction.

```
int let_eval_arg(void) {  
    int      m, p, v;  
  
    m = caddr(Bind_stack);  
    p = Car[m];  
    if (    atomic(p) || Cdr[p] == NIL || atomic(Cdr[p]) ||  
        caddr(p) != NIL || !symbolic(Car[p])  
    ) {  
        /* Error, get rid of the partial environment. */  
        v = bunsave(1);  
        bunsave(3);  
        bsave(v);  
        Env_stack = unsave(1);  
        save(Function_name);  
        save(Frame);  
        unbind_args();  
        return error("let/letrec: bad binding", p);  
    }  
    Car[Bind_stack] = alloc(Car[p], Car[Bind_stack]);  
    return cadr(p);  
}
```

Reverse a list in situ (overwriting the original list).

```
int reverse_in_situ(int n) {  
    int      this, next, x;  
  
    if (n == NIL) return NIL;  
    this = n;  
    next = Cdr[n];  
    Cdr[this] = NIL;  
    while (next != NIL) {  
        x = Cdr[next];  
        Cdr[next] = this;  
        this = next;  
        next = x;  
    }  
    return this;  
}
```

zen style programming

Establish bindings of **let/letrec**. Save outer values on Stack.

```
void let_bind(int env) {
    int    b;

    while (env != NIL) {
        b = Car[env];
        save(cdar(b));          /* Save old value */
        cdar(b) = Cdr[b];      /* Bind new value */
        env = Cdr[env];
    }
}
```

Finish creation of local bindings by **let/letrec**. First load the context from Bind_stack into local variables and clean up Bind_stack. Then perform the actual bindings, saving outer values on Stack. Save a list of local symbols on Bind_stack so that they can be restored later. If this function processes a **letrec** special form (**rec set**), fix cached lexical environments. Finally return the term of the binding construct for further reduction.

This function leaves the same kind of call frame as **lambda** on Bind_stack and Stack.

```
int let_finish(int rec) {
    int    m, v, b, e;

    Tmp2 = alloc(NIL, NIL); /* Create safe storage */
    Cdr[Tmp2] = alloc(NIL, NIL);
    caddr(Tmp2) = alloc(NIL, NIL);
    cdddr(Tmp2) = alloc(NIL, NIL);
    v = bunsave(1);
    b = bunsave(1);          /* bindings */
    m = bunsave(2);          /* drop environment, get full LET/LETREC */
    b = reverse_in_situ(b); /* needed for UNBINDARGS() */
    e = unsave(1);
    Car[Tmp2] = b;
    cadr(Tmp2) = m;
    caddr(Tmp2) = v;
    cdddr(Tmp2) = e;
    let_bind(b);
    bsave(v);
    if (rec) fix_cached_closures();
    Env_stack = e;
    save(Function_name);
    save(Frame);
    Tmp2 = NIL;
    return caddr(m); /* term */
}
```

(let ((symbol expression₁) ...) expression_n) → form

```
int z_let(int n, int *pcf, int *pmode, int *pcbn) {
    *pcf = 2;
```

```
*pmode = MBIND;
USE(pcbn);
if (let_setup(n) != NIL)
    return let_eval_arg();
else
    return NIL;
}
```

(letrec ((symbol expression₁) ...) expression_n) → form

```
int z_letrec(int n, int *pcf, int *pmode, int *pcbn) {
    int    m;

    *pcf = 2;
    *pmode = MBINR;
    USE(pcbn);
    if (let_setup(n) != NIL)
        m = let_eval_arg();
    else
        m = NIL;
    Env_stack = S_true;
    return m;
}
```

(or expression ...) → form

```
int z_or(int n, int *pcf, int *pmode, int *pcbn) {
    USE(pcbn);
    if (Cdr[n] == NIL) {
        return S_false;
    }
    else if (cddr(n) == NIL) {
        *pcf = 1;
        return cadr(n);
    }
    else {
        *pcf = 2;
        *pmode = MDISJ;
        return setup_and_or(n);
    }
}
```

(quote form) → form

```
int z_quote(int n, int *pcf, int *pmode, int *pcbn) {
    int    m;

    USE(pcf);
    USE(pmode);
    USE(pcbn);
    m = Cdr[n];
    if (m == NIL || Cdr[m] != NIL) return wrong_arg_count(n);
}
```


zen style programming

```
        return (Car[m]);
    }

    (closure-form :t | :f) → :t | :f

int z_closure_form(int n, int *pcf, int *pmode, int *pcbn) {
    int
        m;

    USE(pcf);
    USE(pmode);
    USE(pcbn);
    m = Cdr[n];
    if (m == NIL || Cdr[m] != NIL) return wrong_arg_count(n);
    if (!symbolic(Car[m]))
        return error("closure-form: got non-symbol", Car[m]);
    if (Car[m] == add_symbol("args", S_void))
        Closure_form = 0;
    else if (Car[m] == add_symbol("body", S_void))
        Closure_form = 1;
    else if (Car[m] == add_symbol("env", S_void))
        Closure_form = 2;
    else
        return S_false;
    return Car[m];
}
```

These variables are saved when dumping an image file.

```
int *Image_vars[] = {
    &Closure_form, &Verify_arrows,
    &Symbols, &Freelist, &S_bottom, &S_closure, &S_false,
    &S_lambda, &S_primitive, &S_quote, &S_special,
    &S_special_cbv, &S_true, &S_void, &S_last,
    NULL };
};
```

Write a node pool image to the given file. When the image cannot be created or written successfully, report an error.

```
int dump_image(char *p) {
    int    fd, n, i;
    int    **v;
    char    magic[17];

    fd = open(p, O_CREAT | O_WRONLY, 0644);
    setmode(fd, O_BINARY);
    if (fd < 0) {
        error("cannot create file", NO_EXPR);
        Error.arg = p;
        return -1;
    }
    strcpy(magic, "ZEN_____");
    magic[7] = sizeof(int);
```

```

magic[8] = VERSION;
n = 0x12345678;
memcpy(&magic[10], &n, sizeof(int));
write(fd, magic, 16);
n = Pool_size;
write(fd, &n, sizeof(int));
v = Image_vars;
i = 0;
while (v[i]) {
    write(fd, v[i], sizeof(int));
    i = i+1;
}
if (    write(fd, Car, Pool_size*sizeof(int))
      != Pool_size*sizeof(int) ||
      write(fd, Cdr, Pool_size*sizeof(int))
      != Pool_size*sizeof(int) ||
      write(fd, Tag, Pool_size) != Pool_size
) {
    error("dump failed", NO_EXPR);
    close(fd);
    return -1;
}
close(fd);
return 0;
}

```

(dump-image symbol) → :t

```

int z_dump_image(int n, int *pcf, int *pmode, int *pcbn) {
    int      m;
    static char buf[SYMBOL_LEN], *s;

    USE(pcf);
    USE(pmode);
    USE(pcbn);
    m = Cdr[n];
    if (m == NIL || Cdr[m] != NIL) return wrong_arg_count(n);
    if (!symbolic(Car[m]))
        return error("dump-image: got non-symbol",
                     Car[m]);
    s = symbol_to_string(Car[m], buf, SYMBOL_LEN);
    if (s) dump_image(s);
    return S_true;
}

```

The following functions are used by the **load** special form. `Get_source_dir()` extracts the directory part of a file path into a buffer. When the file path has no directory part, the buffer is filled with ".".

```

void get_source_dir(char *path, char *buf) {
    char *p;

```

zen style programming

```
    if (strlen(path) > 256) {
        error("load: path too long", NO_EXPR);
        return;
    }
    strcpy(buf, path);
    p = strrchr(buf, '/');
    if (p == NULL)
        strcpy(buf, ".");
    else
        *p = 0;
}
```

Expand path names beginning with "~" by copying the path to a buffer and replacing the tilde in the copy with \$ZENSRC/ (the value of the ZENSRC environment variable and a slash). When ZENSRC is undefined, simply return the original path.

```
/* Expand leading ~ in path names */
char *expand_path(char *s, char *buf) {
    char    *r, *v;

    if (s[0] == '~')
        r = &s[1];
    else
        return s;
    if ((v = getenv("ZENSRC")) == NULL) return s;
    if (strlen(v) + strlen(r) + 4 >= MAX_PATH_LEN) {
        error("load: path too long", NO_EXPR);
        return s;
    }
    sprintf(buf, "%s/%s", v, r);
    return buf;
}
```

Load a `zenlisp` source file. Read expressions from the file and pass them to the evaluator. `Load()` uses `Load_level` keeps track of nested **loads**. When loading a file from within a file, the same source path will be used for nested **loads**.

Note: the approach used here is buggy, because it does not restore the source path when leaving a directory. Feel free to fix this.

```
int load(char *p) {
    FILE    *ofile, *nfile;
    int      r;
    char     *oname;
    char     *arg;
    int      oline;

    arg = p;
    if (Load_level > 0) {
        if (strlen(p) + strlen(Source_dir) + 4 >= MAX_PATH_LEN) {
```

```

        error("load: path too long", NO_EXPR);
        return -1;
    }
    if (*p != '.' && *p != '/' && *p != '~')
        sprintf(Current_path, "%s/%s", Source_dir, p);
    else
        strcpy(Current_path, p);
    p = Current_path;
}
p = expand_path(p, Expanded_path);
get_source_dir(p, Source_dir);
strcat(p, ".l");
if ((nfile = fopen(p, "r")) == NULL) {
    error("cannot open source file", NO_EXPR);
    Error.arg = arg;
    return -1;
}
Load_level = Load_level + 1;
/* Save I/O state and redirect */
r = Rejected;
ofile = Input;
Input = nfile;
oline = Line;
Line = 1;
oname = Infile;
Infile = p;
read_eval_loop();
Infile = oname;
Line = oline;
/* Restore previous I/O state */
Rejected = r;
Input = ofile;
Load_level = Load_level - 1;
fclose(nfile);
if (Paren_level) error("unbalanced parentheses in loaded file",
                        NO_EXPR);

return 0;
}

```

(load symbol) → :t

```

int z_load(int n, int *pcf, int *pmode, int *pcbn) {
    int    m;
    char    buf[SYMBOL_LEN+1], *s;

    USE(pcf);
    USE(pmode);
    USE(pcbn);
    m = Cdr[n];
    if (m == NIL || Cdr[m] != NIL) return wrong_arg_count(n);
    if (!symbolic(Car[m])) return error("load: got non-symbol", Car[m]);
}

```

zen style programming

```

    s = symbol_to_string(Car[m], buf, SYMBOL_LEN);
    if (s) {
        s = strdup(s);
        if (s == NULL) fatal("load: strdup() failed");
        load(s);
        free(s);
    }
    return S_true;
}

```

(stats expression) → ' (form reductions allocations collections)

```

int z_stats(int n, int *pcf, int *pmode, int *pcbn) {
    int      m;
    char     buf[100];

    USE(pcf);
    USE(pmode);
    USE(pcbn);
    m = Cdr[n];
    if (m == NIL || Cdr[m] != NIL) return wrong_arg_count(n);
    reset_counter(&Allocations);
    reset_counter(&Reductions);
    reset_counter(&Collections);
    Stat_flag = 1;
    n = eval(Car[m]);
    Stat_flag = 0;
    n = alloc(n, NIL);
    save(n);
    Cdr[n] = alloc(NIL, NIL);
    cadr(n) = explode_string(counter_to_string(&Reductions, buf));
    caddr(n) = alloc(NIL, NIL);
    caddr(n) = explode_string(counter_to_string(&Allocations, buf));
    caddr(n) = alloc(NIL, NIL);
    caddr(n) = explode_string(counter_to_string(&Collections, buf));
    unsave(1);
    return n;
}

```

(trace symbol) → :t

(trace) → :t

```

int z_trace(int n, int *pcf, int *pmode, int *pcbn) {
    int      m;
    static char buf[SYMBOL_LEN], *s;

    USE(pcf);
    USE(pmode);
    USE(pcbn);
    m = Cdr[n];
    if (m == NIL) {

```

```

        Traced_fn = NIL;
        return S_true;
    }
    if (Cdr[m] != NIL) return wrong_arg_count(n);
    if (!symbolic(Car[m])) return error("trace: got non-symbol", Car[m]);
    s = symbol_to_string(Car[m], buf, SYMBOL_LEN);
    if (!s) return S_false;
    Traced_fn = find_symbol(s);
    return S_true;
}

```

If `Car[np[0]]` is a keyword, run the corresponding special form handler, set `np[0]` to the result of the operation, and return 1. If `Car[np[0]]` is not a keyword, return 0.

```

int special(int *np, int *pcf, int *pmode, int *pcbn) {
    int    n, y;
    int     (*op)(int, int *, int *, int *);

    n = np[0];
    y = Car[n];
    if (Error_flag) return 0;
    if (Car[y] == S_special || Car[y] == S_special_cbv)
        op = Specials[cadr(y)];
    else if (symbolic(y) &&
             (cadr(y) == S_special ||
              cadr(y) == S_special_cbv))
    )
        op = Specials[caddr(y)];
    else
        return 0;
    np[0] = (*op)(n, pcf, pmode, pcbn);
    return 1;
}

```

12.10 evaluator

The `bind_args()` function binds the variables of a lambda function (closure) to some actual arguments:

For `i` in `((lambda (v1 ... vb) x) a1 ... an)`,

- add `vi` to `Car[Bind_stack]` (initially empty);
- save the value of `vi` on `Stack`;
- bind `vi` to `ai`.

Also save and update the function name and call frame pointer.

Because shallow binding is used, the outer value of each variable has to be saved before re-binding the variable and restored when the context of a function ceases to exist. This approach may look inefficient, but it makes variable lookup a single indirection.

zen style programming

```
void bind_args(int n, int name) {
    int    fa,      /* formal arg list */
           aa,      /* actual arg list */
           e;       /* term */
    int    env;     /* optional lexical environment */
    int    p;
    int    at;      /* atomic argument list flag */

    if (Error_flag) return;
    fa = cadar(n);
    at = symbolic(fa);
    aa = Cdr[n];
    p = cddar(n);
    e = Car[p];
    env = Cdr[p] != NIL ? cadr(p): NIL;
    bsave(NIL); /* names */
    while ((fa != NIL && aa != NIL) || at) {
        if (!at) {
            Car[Bind_stack] = alloc(Car[fa], Car[Bind_stack]);
            save(cdar(fa));
            cdar(fa) = Car[aa];
            fa = Cdr[fa];
            aa = Cdr[aa];
        }
        if (symbolic(fa)) {
            Car[Bind_stack] = alloc(fa, Car[Bind_stack]);
            save(Cdr[fa]);
            Cdr[fa] = aa;
            fa = NIL;
            aa = NIL;
            break;
        }
    }
    while (env != NIL) {
        p = Car[env];
        Car[Bind_stack] = alloc(Car[p], Car[Bind_stack]);
        save(cdar(p));
        cdar(p) = Cdr[p];
        env = Cdr[env];
    }
    if (fa != NIL || aa != NIL) {
        wrong_arg_count(n);
        n = NIL;
    }
    else {
        n = e;
    }
    save(Function_name);
    Function_name = name;
    save(Frame);
    Frame = Stack;
}
```

Print a call to a function being traced:

+ (function argument ...)

```
void print_trace(int n) {
    pr("+ ");
    pr("(");
    Quotedprint = 1;
    print(Traced_fn);
    while (1) {
        n = Cdr[n];
        if (n == NIL) break;
        pr(" ");
        print(Car[n]);
    }
    pr(")"); nl();
}
```

The `eliminate_tail_calls()` function examines the `Mode_stack` to find out whether the caller of the current function in MBETA state. If it is, the call to the current function was a tail call. In this case, `eliminate_tail_calls()` removes all **let**, **letrec**, and **lambda** frames of the caller from `Stack` and `Mode_stack`.

```
void eliminate_tail_calls(void) {
    int m, y;

    m = Car[Mode_stack];
    /* Skip over callee's local frames, if any */
    while (m != NIL && Car[m] == MLETR) {
        m = Cdr[m];
    }
    /* Parent not beta-reducing? Give up. */
    if (m == NIL || Car[m] != MBETA)
        return;
    /* Yes, this is a tail call: */
    /* remove callee's frames. */
    while (1) {
        Tmp2 = unsave(1); /* M */
        unbind_args();
        unsave(1);
        y = munsave();
        save(Tmp2);
        Tmp2 = NIL;
        if (y == MBETA) break;
    }
}
```

The `eval()` function is the heart of the `zenlisp` interpreter. It reduces an expression to its normal form and returns it. The function is guaranteed to run in constant space when the program evaluated by it runs in constant space.

zen style programming

Because `eval()` is a bit long (multiple pages), commentary text is interspersed in the function. The first block of code saves and resets current interpreter state.

```
int eval(int n) {
    int      m,          /* Result node */
            m2,         /* Root of result lists */
            a;          /* Used to append to result */
    int      mode,       /* Current state */
            cf,         /* Continue flag */
            cbn;        /* Call by name flag */
    int      nm;         /* Name of function to apply */

    Eval_level = Eval_level + 1;
    save(n);
    save(Arg_stack);
    save(Bind_stack);
    save(Car[Mode_stack]);
    save(Stack_bottom);
    Stack_bottom = Stack;
    mode = MATOM;
    cf = 0;
    cbn = 0;
```

In the following loop, **n** holds a source expression (which may have been generated by a special form) and **m** finally holds the corresponding normal form.

```
while (!Error_flag) {
    if (Stat_flag) count(&Reductions, 1);
    if (n == NIL) {
        m = NIL;
        cbn = 0;
    }
    else if (symbolic(n)) {
        /* Symbol -> Value */
        if (cbn) {
            m = n;
            cbn = 0;
        }
        else {
            m = Cdr[n] == Car[n]? n: Cdr[n];
            if (m == S_void) {
                error("symbol not bound", n);
                break;
            }
        }
    }
}
```

Hack alert! When the **cbn** “flag” is set to 2, this means “do not evaluate this expression at all” rather than just “call by name”. Hence **cbn==2** may be considered a “stronger” form of **cbn==1**.

```
    else if (Car[n] == S_closure ||
            Car[n] == S_primitive ||
```

```

        Car[n] == S_special ||
        Car[n] == S_special_cbv ||
        cbn == 2
    ) {
        m = n;
        cbn = 0;
    }

```

The following branch is used to descend into a list. It saves various values on the stacks for processing by the loop that follows the branch. The saved values are:

- the original source list (on Stack);
- the current state (on Mode_stack);
- the result list (filled by the subsequent loop, on Arg_stack);
- a pointer for appending values to the result list (on Arg_stack);
- the remaining members to evaluate (on Arg_stack).

When call by value is used, the result list will be initialized with `()` and the append pointer `a` will point to the same form. To append a normal form `x`, it is then sufficient to store it in `Car[a]`. Then a new empty list is stored in `Cdr[a]` and `a` advances to the `cdr` part. So appending elements is an $O(1)$ operation.

When call by name is used, the result list is a copy of the source list and the remaining member list is set to `()`.

```

    else {
        /* List (...) and Pair (X.Y) */
        m = Car[n];
        save(n);
        msave(mode);
        if ((symbolic(m) && cadr(m) == S_special) || cbn) {
            cbn = 0;
            asave(NIL);
            asave(NIL);
            asave(n); /* Root of result list */
            n = NIL;
        }
        else {
            a = alloc(NIL, NIL);
            asave(a);
            asave(Cdr[n]);
            asave(a); /* Root of result list */
            n = Car[n];
        }
        mode = MLIST;
        continue;
    }

```

The following loop evaluates the members of a list, performs function applications and reduces special forms. Note that the indentation of the `while` loop is wrong. The loop body spans more

zen style programming

than 100 lines. You will find a remainder at the end of the loop.

In MBETA state, all that is left to do is to clean up the context of the calling function and return to the outer list (if any).

```
while (1) if (mode == MBETA || mode == MLETR) {
    /* Finish BETA reduction */
    unbind_args();
    unsave(1);
    mode = munsave();
}
```

In MLIST mode, members of a list are reduced to their normal forms.

```
else if (mode == MLIST) {
    n = cadr(Arg_stack);      /* Next member */
    a = caddr(Arg_stack);     /* Place to append to */
    m2 = aunsave(1);          /* Root of result list */
```

OK, got a complete list, now decide what do do with it.

```
if (a != NIL) Car[a] = m;
if (n == NIL) {              /* End of list */
    m = m2;
    aunsave(2);              /* Drop N,A */
    nm = Car[unsave(1)];
    save(m);                 /* Save result */
    if (Traced_fn == nm) print_trace(m);
    if (primitive(&m))
        ;
    else if (special(&m, &cf, &mode, &cbn))
        n = m;
    else if (!atomic(Car[m]) &&
              caar(m) == S_closure
            ) {
        nm = symbolic(nm)? nm: NIL;
        eliminate_tail_calls();
        bind_args(m, nm);
        /* N=E of ((LAMBDA (...) E) ...) */
        n = caddr(m);
        cf = 2;
        mode = MBETA;
    }
    else {
        error("application of non-function",
              nm);
        n = NIL;
    }
}
```

Another “flag” hack. **Cf==2** means that the current context cannot be abandoned yet, because the current evaluation is still in progress. This happens only when evaluating terms of lambda

functions. In this case, the clean up is performed by setting the mode=MBETA.

```
        if (cf != 2) {
            unsave(1);
            mode = munsave();
        }
        /* Leave the list loop and re-evaluate N */
        if (cf) break;
    }
```

End of list not yet reached, insert current member, append new empty slot, and prepare next member for evaluation.

```
    else {                /* N != NIL: Append to list */
        asave(m2);
        Cdr[a] = alloc(NIL, NIL);
        caddr(Arg_stack) = Cdr[a];
        cadr(Arg_stack) = Cdr[n];
        if (symbolic(n))
            error("improper list in application",
                n);
        n = Car[n];        /* Evaluate next member */
        break;
    }
}
```

This is the place where the binding constructs and control flow constructs are handled. This is still part of the list evaluating loop.

This branch evaluates **cond** expressions.

```
    else if (mode == MCOND) {
        n = cond_eval_clause(m);
        if (Car[Bind_stack] == NIL) {
            unsave(1);
            bunsave(1);
            mode = munsave();
        }
        cf = 1;
        break;
    }
```

Evaluate **and** and **or**.

```
    else if (mode == MCONJ || mode == MDISJ) {
        Car[Bind_stack] = cdar(Bind_stack);
        if ( (m == S_false && mode == MCONJ) ||
            (m != S_false && mode == MDISJ) ||
            Car[Bind_stack] == NIL
        ) {
            unsave(1);
            bunsave(1);
        }
    }
```

zen style programming

```
        mode = munsave();
        n = m;
        cbn = 2;
    }
    else if (cdar(Bind_stack) == NIL) {
        n = caar(Bind_stack);
        unsave(1);
        bunsave(1);
        mode = munsave();
    }
    else {
        n = caar(Bind_stack);
    }
    cf = 1;
    break;
}
```

Evaluate **let** and **letrec**.

```
    else if (mode == MBIND || mode == MBINR) {
        if (let_next_binding(m) == NIL) {
            n = let_finish(mode == MBINR);
            mode = MLETR;
        }
        else {
            n = let_eval_arg();
        }
        cf = 1;
        break;
    }
```

If the expression to evaluate was an atom, there is nothing left to do.

```
    else { /* Atom */
        break;
    }
```

The list evaluating loop ends above.

```
        if (cf) { /* Continue evaluation if requested */
            cf = 0;
            continue;
        }
        if (Stack == Stack_bottom) break;
    }
```

Restore the previous state of the interpreter.

```
    while (Stack != Stack_bottom) unsave(1);
    Stack_bottom = unsave(1);
    Car[Mode_stack] = unsave(1);
    Bind_stack = unsave(1);
```

```
    Arg_stack = unsave(1);
    unsave(1);
    Eval_level = Eval_level - 1;
    return m;
}
```

12.11 printer

The helper functions that precede `print()` all work in the same way. They check whether their argument has a specific property and if it has that property, they print the argument and return 1. Otherwise they return 0.

Print (**quote x**) as '**x**'.

```
int print_quoted_form(int n, int dot) {
    if (    Car[n] == S_quote &&
          Cdr[n] != NIL &&
          caddr(n) == NIL
        ) {
        if (dot) pr(" . ");
        n = cadr(n);
        if (n != S_true && n != S_false) pr("'");
        print(n);
        return 1;
    }
    return 0;
}
```

Print lists of single-character symbols as condensed lists.

```
int print_condensed_list(int n, int dot) {
    int    m;
    char    s[2];

    m = n;
    if (m == NIL) return 0;
    while (m != NIL) {
        if (!symbolic(Car[m])) return 0;
        if (cdaar(m) != NIL) return 0;
        m = Cdr[m];
    }
    if (dot) pr(" . ");
    pr("#");
    m = n;
    s[1] = 0;
    while (m != NIL) {
        s[0] = caaar(m);
        pr(s);
        m = Cdr[m];
    }
}
```

zen style programming

```
        return 1;
    }
}
```

Print closures. The `Closure_form` variable determines the amount of information to print. Note that closures are ambiguous when no environment is printed. Hence this function prints **{closure ...}** instead of **(closure ...)** when `Closure_form` is less than 2.

```
int print_closure(int n, int dot) {
    if (    Car[n] == S_closure &&
          !atomic(Cdr[n]) &&
          !atomic(caddr(n))
        ) {
        Quotedprint = 1;
        if (dot) pr(" . ");
        pr(Closure_form==2? "(closure ": "{closure ");
        print(cadr(n));
        if (Closure_form > 0) {
            pr(" ");
            print(caddr(n));
            if (Closure_form > 1 && caddr(n) != NIL) {
                pr(" ");
                print(caddr(n));
            }
        }
        pr(Closure_form==2? "": "}");
        return 1;
    }
    return 0;
}
```

Print primitive function handlers and special form handlers.

```
int print_primitive(int n, int dot) {
    if (    Car[n] != S_primitive &&
          Car[n] != S_special &&
          Car[n] != S_special_cbv
        )
        return 0;
    if (dot) pr(" . ");
    pr("{internal ");
    Quotedprint = 1;
    print(caddr(n));
    pr("}");
    return 1;
}
```

This is the `zenlisp` printer interface. It converts the internal node representation of a form into its external (human-readable) form and emits it.

```
void print(int n) {
    char    s[SYMBOL_LEN+1];
```

```
int      i;

if (n == NIL) {
    pr("(");
}
else if (n == S_void) {
    pr("{void}");
}
else if (Tag[n] & ATOM_FLAG) {
    /* Characters are limited to the symbol table */
    pr("{unprintable form}");
}
else if (symbolic(n)) {
    if (!Quotedprint && n != S_true && n != S_false) {
        pr("'");
        Quotedprint = 1;
    }
    i = 0;          /* Symbol */
    n = Car[n];
    while (n != NIL) {
        s[i] = Car[n];
        if (i > SYMBOL_LEN-2) break;
        i += 1;
        n = Cdr[n];
    }
    s[i] = 0;
    pr(s);
}
else { /* List */
    if (print_closure(n, 0)) return;
    if (print_primitive(n, 0)) return;
    if (!Quotedprint) {
        pr("'");
        Quotedprint = 1;
    }
    if (print_quoted_form(n, 0)) return;
    if (print_condensed_list(n, 0)) return;
    pr("(");
    while (n != NIL) {
        print(Car[n]);
        n = Cdr[n];
        if (symbolic(n) || n == S_void) {
            pr(" . ");
            print(n);
            n = NIL;
        }
        if (print_closure(n, 1)) break;
        if (print_primitive(n, 1)) break;
        if (print_quoted_form(n, 1)) break;
        if (n != NIL) pr(" ");
    }
}
```


zen style programming

```
        pr(" ");
    }
}
```

12.12 initialization

Reset the state of the interpreter: clear the stacks and debugging variables and reset the level counters.

```
void reset_state(void) {
    Stack = NIL;
    Arg_stack = NIL;
    Bind_stack = NIL;
    Env_stack = NIL;
    Frame = NIL;
    Function_name = NIL;
    Eval_level = 0;
    Paren_level = 0;
}
```

First stage of interpreter initialization. Initialize miscellaneous variables, clear the free list, connect the input/output streams.

```
void init1() {
    /* Misc. variables */
    reset_state();
    Mode_stack = NIL;
    Error_flag = 0;
    Error.arg = NULL;
    Fatal_flag = 0;
    Symbols = NIL;
    Safe_symbols = NIL;
    Tmp_car = NIL;
    Tmp_cdr = NIL;
    Tmp = NIL;
    Tmp2 = NIL;
    Load_level = 0;
    Traced_fn = NIL;
    Max_atoms_used = 0;
    Max_trace = 10;
    Stat_flag = 0;
    Closure_form = 0;
    Verify_arrows = 0;
    Line = 1;
    /* Initialize Freelist */
    Freelist = NIL;
    /* Clear input buffer */
    Infile = NULL;
    Source_dir[0] = 0;
    Input = stdin;
    Output = stdout;
}
```

```
    Rejected = EOT;
}
```

Second stage of interpreter initialization: build free list, create built-in symbols.

```
void init2(void) {
    /*
     * Tags (especially 'primitive and 'special*)
     * must be defined before the primitives.
     * First GC will be triggered HERE
     */
    S_void = add_symbol("{void}", 0);
    S_special = add_symbol("{special}", 0);
    S_special_cbv = add_symbol("{special/cbv}", 0);
    S_primitive = add_symbol("{primitive}", 0);
    S_closure = add_symbol("closure", 0);
    add_primitive("atom", P_ATOM);
    add_special("and", SF_AND, 0);
    add_special("apply", SF_APPLY, 1);
    S_bottom = add_primitive("bottom", P_BOTTOM);
    add_primitive("car", P_CAR);
    add_primitive("cdr", P_CDR);
    add_special("closure-form", SF_CLOSURE_FORM, 0);
    add_special("cond", SF_COND, 0);
    add_primitive("cons", P_CONS);
    add_special("define", SF_DEFINE, 0);
    add_primitive("defined", P_DEFINED);
    add_special("dump-image", SF_DUMP_IMAGE, 0);
    add_special("eval", SF_EVAL, 1);
    add_primitive("eq", P_EQ);
    add_primitive("explode", P_EXPLODE);
    S_false = add_symbol(":f", 0);
    add_primitive("gc", P_GC);
    add_primitive("implode", P_IMPLODE);
    S_lambda = add_special("lambda", SF_LAMBDA, 0);
    add_special("let", SF_LET, 0);
    add_special("letrec", SF_LETREC, 0);
    add_special("load", SF_LOAD, 0);
    add_special("or", SF_OR, 0);
    add_primitive("quit", P_QUIT);
    S_quote = add_special("quote", SF_QUOTE, 0);
    add_primitive("recursive-bind", P_RECURSIVE_BIND);
    add_special("stats", SF_STATS, 0);
    add_primitive("symbols", P_SYMBOLS);
    S_true = add_symbol(":t", 0);
    add_symbol("t", S_true);
    add_special("trace", SF_TRACE, 0);
    add_primitive("verify-arrows", P_VERIFY_ARROWS);
    S_last = add_symbol("***", 0);
    Mode_stack = alloc(NIL, NIL);
    Primitives[P_ATOM] = &z_atom;
```

zen style programming

```
Primitives[P_BOTTOM] = &z_bottom;
Primitives[P_CAR] = &z_car;
Primitives[P_CDR] = &z_cdr;
Primitives[P_CONS] = &z_cons;
Primitives[P_DEFINED] = &z_defined;
Primitives[P_EQ] = &z_eq;
Primitives[P_EXPLODE] = &z_explode;
Primitives[P_GC] = &z_gc;
Primitives[P_IMPLODE] = &z_implode;
Primitives[P_QUIT] = &z_quit;
Primitives[P_RECURSIVE_BIND] = &z_recursive_bind;
Primitives[P_SYMBOLS] = &z_symbols;
Primitives[P_VERIFY_ARROWS] = &z_verify_arrows;
Specials[SF_AND] = &z_and;
Specials[SF_APPLY] = &z_apply;
Specials[SF_CLOSURE_FORM] = &z_closure_form;
Specials[SF_COND] = &z_cond;
Specials[SF_DEFINE] = &z_define;
Specials[SF_DUMP_IMAGE] = &z_dump_image;
Specials[SF_EVAL] = &z_eval;
Specials[SF_LAMBDA] = &z_lambda;
Specials[SF_LET] = &z_let;
Specials[SF_LETREC] = &z_letrec;
Specials[SF_LOAD] = &z_load;
Specials[SF_OR] = &z_or;
Specials[SF_QUOTE] = &z_quote;
Specials[SF_STATS] = &z_stats;
Specials[SF_TRACE] = &z_trace;
}
```

Clear **stats** counters.

```
void clear_stats(void) {
    reset_counter(&Reductions);
    reset_counter(&Allocations);
    reset_counter(&Collections);
}
```

12.13 interpreter interface

Load a node pool image from a given file. Return zero upon success and a non-zero value in case of an error.

```
int zen_load_image(char *p) {
    int    fd, n, i;
    char   buf[17];
    int    **v;
    int    bad = 0;
    int    inodes;
```

```
fd = open(p, O_RDONLY);
setmode(fd, O_BINARY);
if (fd < 0) {
    error("cannot open image", NO_EXPR);
    Error.arg = p;
    return -1;
}
memset(Tag, 0, Pool_size);
read(fd, buf, 16);
if (memcmp(buf, "ZEN____", 7)) {
    error("bad image (magic match failed)", NO_EXPR);
    bad = 1;
}
if (buf[7] != sizeof(int)) {
    error("bad image (wrong cell size)", NO_EXPR);
    bad = 1;
}
if (buf[8] != VERSION) {
    error("bad image (wrong version)", NO_EXPR);
    bad = 1;
}
memcpy(&n, &buf[10], sizeof(int));
if (n != 0x12345678) {
    error("bad image (wrong architecture)", NO_EXPR);
    bad = 1;
}
read(fd, &inodes, sizeof(int));
if (inodes > Pool_size) {
    error("bad image (too many nodes)", NO_EXPR);
    bad = 1;
}
v = Image_vars;
i = 0;
while (v[i]) {
    read(fd, v[i], sizeof(int));
    i = i+1;
}
if ( !bad &&
    (read(fd, Car, inodes*sizeof(int)) != inodes*sizeof(int) ||
     read(fd, Cdr, inodes*sizeof(int)) != inodes*sizeof(int) ||
     read(fd, Tag, inodes) != inodes)
) {
    error("bad image (bad file size)", NO_EXPR);
    bad = 1;
}
close(fd);
if (bad) Error.arg = p;
return Error_flag;
}
```

zen style programming

Main initialization. Allocate node pool, clear tags, initialize variables.

```
int zen_init(int nodes, int vgc) {
    Pool_size = nodes? nodes: DEFAULT_NODES;
    Verbose_GC = vgc;
    if (Pool_size < MINIMUM_NODES) return -1;
    if ( (Car = (int *) malloc(Pool_size * sizeof(int))) == NULL ||
        (Cdr = (int *) malloc(Pool_size * sizeof(int))) == NULL ||
        (Tag = (char *) malloc(Pool_size)) == NULL
    ) {
        if (Car) free(Car);
        if (Cdr) free(Cdr);
        if (Tag) free(Tag);
        Car = Cdr = NULL;
        Tag = NULL;
        return -1;
    }
    memset(Tag, 0, Pool_size);
    init1();
    init2();
    return 0;
}
```

De-allocate the node pools.

```
void zen_fini() {
    if (Car) free(Car);
    if (Cdr) free(Cdr);
    if (Tag) free(Tag);
    Car = Cdr = NULL;
    Tag = NULL;
}
```

Stop the interpreter (after receiving an “interrupt” signal (SIGINT) from the user).

```
void zen_stop(void) {
    error("interrupted", NO_EXPR);
}
```

I/O interface.

```
void zen_print(int n) {
    Quotedprint = 0;
    print(n);
}

int zen_read(void) {
    Paren_level = 0;
    return zread();
}
```

Create a copy of the symbol table. This copy will be kept in a safe location (Safe_symbols), so

it can be used to restore the symbol table in case something goes totally wrong.

```
int copy_bindings(void) {
    int    y, p, ny, q;

    p = alloc(NIL, NIL);
    save(p);
    ny = p;
    q = NIL;
    y = Symbols;
    while (y != NIL) {
        Car[p] = alloc(Car[y], cdr(y));
        y = Cdr[y];
        Cdr[p] = alloc(NIL, NIL);
        q = p;
        p = Cdr[p];
    }
    if (q != NIL) Cdr[q] = NIL;
    unsave(1);
    return Car[ny] == NIL? NIL: ny;
}
```

Restore bindings saved by copy_bindings().

```
void restore_bindings(int values) {
    int    b;

    while (values != NIL) {
        b = Car[values];
        cdr(b) = Cdr[b];
        values = Cdr[values];
    }
}
```

Safely reduce an expression to its normal form. Bail out gracefully in case of an error.

```
int zen_eval(int n) {
    save(n);
    Safe_symbols = copy_bindings();
    if (Stat_flag) clear_stats();
    n = eval(n, 0);
    unsave(1);
    if (!Error_flag) {
        Cdr[S_last] = n;
        if (Stack != NIL)
            fatal("eval(): unbalanced stack");
    }
    else {
        restore_bindings(Safe_symbols);
    }
    reset_state();
    while (Car[Mode_stack] != NIL) munsave();
}
```

zen style programming

```
        return n;
    }
```

The obligatory license text.

```
char **zen_license() {
    static char    *license_text[] = {
        "",
        "zenlisp -- An interpreter for symbolic LISP",
        "By Nils M Holm, 2007, 2008",
        "",
        "Don't worry, be happy.",
        "",
        "THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND",
        "ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE",
        "IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE",
        "ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE",
        "FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL",
        "DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS",
        "OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)",
        "HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT",
        "LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY",
        "OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF",
        "SUCH DAMAGE.",
        "",
        NULL};
    return license_text;
}
```

This is a simple, internal read-eval loop (without print). It is used for loading programs.

```
void read_eval_loop(void) {
    int      n, evl;

    Error_flag = 0;
    evl = Eval_level;
    Eval_level = 0;
    while(!Error_flag) {
        n = zen_read();
        if (n == EOT) break;
        n = eval(n, 0);
    }
    Eval_level = evl;
}
```

12.14 interpreter shell

Here begins the user interface of the `zenlisp` interpreter. There are only very few connections between the code above and below. Both parts could be placed in different files with few modifications.

This header is required for handling keyboard interrupts:

```
#include <signal.h>
```

Image holds the name of the node pool image to load. Nodes is the size of the node pool to allocate. Batch is a flag indicating whether the interpreter shall run in batch mode. When GC_stats is set to 1, the interpreter will print some information after each garbage collection.

In *batch mode*, the interpreter will

- not print a banner or ==> operators;
- exit immediately after reporting an error;
- exit immediately when catching SIGINT.

```
char    Image[MAX_PATH_LEN];
int      Nodes;
int      Batch;
int      GC_stats;

void usage(void) {
    fprintf(stderr,
            "Usage: zl [-L] [-bgi] [-n nodes] [image]\n");
}
```

Retrieve the (numeric) value associated with a command line option. See `get_options()` for the meanings of **pi**, **pj**, **pk**. Return the retrieved value. Values are normally specified in ones, but a suffix of “K” or “M” may be used to specify “kilos” (1024’s) or “megas” (1024²’s) respectively.

```
int get_opt_val(int argc, char **argv, int *pi, int *pj, int *pk) {
    int    n, c;

    if (++(*pi) >= argc) {
        usage();
        exit(1);
    }
    n = atoi(argv[*pi]);
    c = argv[*pi][strlen(argv[*pi])-1];
    switch (c) {
    case 'K':      n = n * 1024; break;
    case 'M':      n = n * 1024 * 1024; break;
    }
    *pj = *pk = 0;
    return n;
}

void help(void) {
    fputc('\n', stderr);
    usage();
    fprintf(stderr,
            "\n"
```


zen style programming

```
        "-b    batch mode (quiet, exit on first error)\n"
        "-g    report number of free nodes after each GC\n"
        "-i    init mode (do not load any image)\n"
        "-n #   number of nodes to allocate (default: %dK)\n"
        "-L    print license and exit\n"
        "\n"
        "default image: %s\n\n",
        DEFAULT_NODES/1024, DEFAULT_IMAGE);
}

void print_license(void) {
    char    **s;

    s = zen_license();
    while (*s) {
        printf("%s\n", *s);
        s++;
    }
    exit(0);
}
```

Parse the command line options passed to the interpreter shell. Set default values in case no options are given. The variables **i** (current option), **j** (current character of option string), and **k** (length of current option string) are passed to `get_opt_val()` as pointers in order to extract argument values of options.

```
void get_options(int argc, char **argv) {
    char    *a;
    int     i, j, k;
    int     v;

    strncpy(Image, DEFAULT_IMAGE, strlen(DEFAULT_IMAGE));
    Image[MAX_PATH_LEN-1] = 0;
    Nodes = DEFAULT_NODES;
    GC_stats = 0;
    Batch = 0;
    v = 0;
    i = 1;
    while (i < argc) {
        a = argv[i];
        if (a[0] != '-') break;
        k = strlen(a);
        for (j=1; j<k; j++) {
            switch (a[j]) {
                case 'b':
                    Batch = 1;
                    break;
                case 'n':
                    Nodes = get_opt_val(argc, argv, &i, &j, &k);
                    break;
                case 'g':
```

```
        GC_stats = 1;
        break;
    case 'i':
        Image[0] = 0;
        break;
    case 'L':
        print_license();
        break;
    case '?':
    case 'h':
        help();
        exit(1);
        break;
    default:
        usage();
        exit(1);
    }
    i = i+1;
}
if (i < argc) {
    strncpy(Image, a, strlen(a)+1);
    Image[MAX_PATH_LEN-1] = 0;
}
if (Nodes < MINIMUM_NODES) {
    fprintf(stderr, "zenlisp: minimal pool size is %d\n",
        MINIMUM_NODES);
    exit(1);
}
}
```

SIGINT handler.

```
void catch_int(int sig) {
    USE(sig);
    zen_stop();
    signal(SIGINT, catch_int);
}
```

The `repl()` function implements the main interpreter loop, the so-called REPL (**read-eval-print loop**). As its name suggests, it reads an expression from the input stream, evaluates it, prints the resulting normal form (if any) and finally loops. The REPL exits when it receives an EOT character (or after reporting an error in batch mode).

```
void repl(void) {
    int n;

    while(1) {
        Error_flag = 0;
        n = zen_read();
        if (n == EOT) return;
```

zen style programming

```
        if (Error_flag) {
            zen_print_error();
            if (Batch) exit(1);
            continue;
        }
        n = zen_eval(n);
        if (Error_flag) {
            zen_print_error();
            if (Batch) exit(1);
        }
        else {
            if (!Batch) pr("=> ");
            zen_print(n);
            nl();
        }
    }
}

void init(void) {
    if (zen_init(Nodes, GC_stats)) {
        fprintf(stderr, "zenlisp init failed (memory problem)\n");
        exit(1);
    }
}
```

Ready to lift off...

In case you wonder why options are checked twice: the first pass sets pre-initialization options, the second one post-initialization options.

```
int main(int argc, char **argv) {
    get_options(argc, argv);
    init();
    get_options(argc, argv);
    if (!Batch) {
        pr("zenlisp ");
        pr(RELEASE);
        pr(" by Nils M Holm");
        nl();
    }
    if (Image[0]) {
        if (zen_load_image(Image)) {
            zen_print_error();
            if (Batch) exit(1);
            zen_fini();
            init();
            get_options(argc, argv);
        }
    }
    else if (!Batch) {
        pr("Warning: no image loaded");
    }
}
```

268

```
        nl();
    }
    signal(SIGINT, catch_int);
    repl();
    zen_fini();
    return 0;
}
```

13. lisp part

13.1 base library

This part describes the LISP functions contained in the default image of the `zenlisp` interpreter. They are contained in the file `base.l`. The functions defined here have been discussed in detail in the first part of this book.

Each function definition is preceded by a prototype. Prototypes provide additional, semi-formal information about a function, but they are not part of the `zenlisp` language. See page 222 for an explanation of function prototypes.

Names of functions that are part of the `zenlisp` language print in boldface characters in their definitions. Function names not printed in boldface characters are internal to their packages and should never be used in user-level `zenlisp` code.

```
; zenlisp base functions
; By Nils M Holm, 2007, 2008
; Feel free to copy, share, and modify this code.
; See the file LICENSE for details.
```

```
(define base :t)

(null form) → :t | :f

(define (null x) (eq x ()))

(id form) → form

(define (id x) x)

(list form ...) → list

(define (list . x) x)

(not form) → :t | :f

(define (not a) (eq a :f))

(neq form1 form2) → :t | :f

(define (neq x y) (eq (eq x y) :f))

(caar pair) → form
...
(cdddr pair) → form

(define (caaar x) (car (car (car (car x)))))
(define (caaadr x) (car (car (car (cdr x)))))
```

```
(define (caadar x) (car (car (cdr (car x)))))
(define (caaddr x) (car (car (cdr (cdr x)))))
(define (cadaar x) (car (cdr (car (car x)))))
(define (cadadr x) (car (cdr (car (cdr x)))))
(define (caddar x) (car (cdr (cdr (car x)))))
(define (caddr x) (car (cdr (cdr (cdr x)))))
(define (cdaaar x) (cdr (car (car (car x)))))
(define (cdaadr x) (cdr (car (car (cdr x)))))
(define (cdadar x) (cdr (car (cdr (car x)))))
(define (cdaddr x) (cdr (car (cdr (cdr x)))))
(define (cddaar x) (cdr (cdr (car (car x)))))
(define (cddadr x) (cdr (cdr (car (cdr x)))))
(define (cdddar x) (cdr (cdr (cdr (car x)))))
(define (cdddr x) (cdr (cdr (cdr (cdr x)))))
```

```
(define (caaar x) (car (car (car x))))
(define (caadr x) (car (car (cdr x))))
(define (cadar x) (car (cdr (car x))))
(define (caddr x) (car (cdr (cdr x))))
(define (cdaar x) (cdr (car (car x))))
(define (cdadr x) (cdr (car (cdr x))))
(define (cddar x) (cdr (cdr (car x))))
(define (cddr x) (cdr (cdr (cdr x))))
```

```
(define (caar x) (car (car x)))
(define (cadr x) (car (cdr x)))
(define (cdar x) (cdr (car x)))
(define (cddr x) (cdr (cdr x)))
```

(fold function form list) → form

```
(define (fold f x a)
  (letrec
    ((fold2
      (lambda (a res)
        (cond ((null a) res)
              (t (fold2 (cdr a)
                        (f res (car a)))))))
    (fold2 a x)))
```

(fold-r function form list) → form

```
(define (fold-r f x a)
  (letrec
    ((fold2
      (lambda (a)
        (cond ((null a) x)
              (t (f (car a)
                    (fold2 (cdr a)))))))
    (fold2 a)))
```

zen style programming

(reverse list) → list

```
(define (reverse a)
  (letrec
    ((reverse2
      (lambda (a b)
        (cond ((null a) b)
              (t (reverse2 (cdr a)
                           (cons (car a) b))))))
    (reverse2 a ())))
```

(append list ...) → list

(append atom) → atom

(append list₁ list₂ ... atom) → dotted list

```
(define (append . a)
  (letrec
    ((append2
      (lambda (a b)
        (cond ((null a) b)
              (t (append2 (cdr a) (cons (car a) b))))))
    (fold (lambda (a b) (append2 (reverse a) b))
          ()
          a)))
```

(equal form₁ form₂) → :t | :f

The first clause of **equal** returns truth immediately when comparing identical structures (like shared tails of lists). This is a performance hack.

```
(define (equal a b)
  (cond ((eq a b) :t)
        ((or (atom a) (atom b))
         (eq a b))
        (t (and (equal (car a) (car b))
                  (equal (cdr a) (cdr b))))))
```

(assoc form alist) → pair | :f

```
(define (assoc x a)
  (cond ((null a) :f)
        ((equal (caar a) x) (car a))
        (t (assoc x (cdr a)))))
```

(assq atom alist) → pair | :f

```
(define (assq x a)
  (cond ((null a) :f)
        ((eq (caar a) x) (car a))
        (t (assq x (cdr a)))))
```

(listp form) → :t | :f

```
(define (listp x)
  (or (null x)
      (and (not (atom x))
            (listp (cdr x)))))
```

(map function list₁ list₂ ...) → list

```
(define (map f . a)
  (letrec
    ((map-car
      (lambda (f a r)
        (cond ((null a) (reverse r))
              (t (map-car f (cdr a) (cons (f (car a)) r))))))
    (car-of
      (lambda (a)
        (map-car car a ())))
    (cdr-of
      (lambda (a)
        (map-car cdr a ())))
    (any-null
      (lambda (a)
        (apply or (map-car null a ())))))
    (map2
      (lambda (a b)
        (cond ((any-null a) (reverse b))
              (t (map2 (cdr-of a)
                        (cons (apply f (car-of a)) b))))))
    (cond ((null a) (bottom '(too few arguments to map)))
          (t (map2 a ())))))
```

(member form list) → form | :f

```
(define (member x a)
  (cond ((null a) :f)
        ((equal (car a) x) a)
        (t (member x (cdr a)))))
```

(memq atom list) → form | :f

```
(define (memq x a)
  (cond ((null a) :f)
        ((eq (car a) x) a)
        (t (memq x (cdr a)))))
```

(require symbol) → :t | :f

```
(define (require x)
  (letrec
    ((require2
      (lambda (sym file)
```


zen style programming

```
(cond ((defined sym) :f)
      (t (apply load (list file))))))
(let ((xx (explode x)))
  (cond ((eq (car xx) '~)
        (require2 (implode (cdr xx)) x))
        (t (require2 x x)))))
```

13.2 iterator package

The **iter** package defines iterators for arithmetic function and predicates. These functions have been discussed in great detail in the first part of this book [pages 45 and 53].

The iterator package is contained in the file `iter.l`.

```
; zenlisp iterators
; By Nils M Holm, 2007, 2008
; Feel free to copy, share, and modify this code.
; See the file LICENSE for details.
```

```
(define iter :t)
```

(arithmetic-iterator function₁ function₂ number) → function

```
(define (arithmetic-iterator conv fn neutral)
  (lambda x
    (cond ((null x) neutral)
          (t (fold (lambda (a b)
                     (fn (conv a) (conv b)))
                   (car x)
                   (cdr x))))))
```

(predicate-iterator function₁ function₂) → function

```
(define (predicate-iterator conv fn)
  (let ((fail (cons 'fail ())))
    (let ((comp (lambda (a b)
                  (cond ((eq a fail) fail)
                        ((fn (conv a) (conv b)) b)
                        (t fail)))))
      (lambda (first . rest)
        (cond ((null rest) (bottom '(too few arguments)))
              (t (neq (fold comp first rest) fail))))))
```

13.3 natural math functions

The **nmath** package implements natural number arithmetics on top of the symbols '0...'9. Each natural number is a list of these symbols. This is why numbers can be written in condensed form, e.g. '**#31415**' instead of '**(3 1 4 1 5)**'. In fact, condensed lists were originally invented in order to provide a more convenient notation for numbers. This is also the reason why the "number"

sign was chosen to introduce condensed lists.

The **nmath** package forms the foundation of the numeric tower of **zenlisp** as depicted in figure 17. Each layer of the numeric towers builds on top of the “lower” levels that implement more primitive types.

Nmath itself consists of three layers implementing arithmetic tables, functions that use these tables to implement single-digit arithmetics and, finally, the natural math functions themselves.

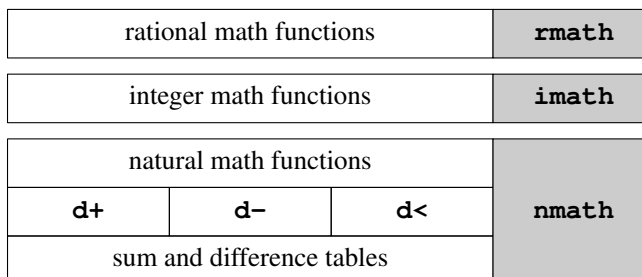


Fig. 17 – numeric tower

Each of the math packages can be loaded individually, but loading **imath** will include **nmath**, and loading **rmath** will load the complete numeric tower.

While the more primitive math packages lack some functionality that the more complex packages provide, they are often preferable because they are much more efficient when performing the same task.

For example, computing 2^{100} using natural arithmetics is *much* faster than using the corresponding rational math function. The natural and integer versions, on the other hand, do not accept negative exponents. Hence

Math packages should be chosen carefully.

The natural math package is contained in the file `nmath.l`.

```
; zenlisp natural math functions
; By Nils M Holm, 2007
; Feel free to copy, share, and modify this code.
; See the file LICENSE for details.
```

Nmath requires **base**, but **require** is defined in **base**, so it cannot be used here:

```
(cond ((defined 'base) :f)
      (t (load base)))

(define nmath :t)
```

zenstyle programming

First define the digit symbols as constants, so you can write 0 instead of '0.

```
(define 0 '0)
(define 1 '1)
(define 2 '2)
(define 3 '3)
(define 4 '4)
(define 5 '5)
(define 6 '6)
(define 7 '7)
(define 8 '8)
(define 9 '9)

(define *digits* '#0123456789)
```

(digitp form) → :t | :f

```
(define (digitp x) (and (memq x *digits*) :t))
```

The **succ** and **pred** functions compute the successor and predecessor of a digit. Both of them return **:f** when an overflow or underflow occurs. The functions are not used much in the code. **Pred** is used only a few times and **succ** is not used at all (it is kept for reasons of symmetry, though). Both functions were involved in the implementation of more complex numeric functions in earlier versions (ArrowLISP), but zenlisp uses table-driven arithmetics instead. See below for details.

(succ symbol) → symbol | :f

```
(define (succ x)
  (cond ((eq x 0) 1)
        ((eq x 1) 2)
        ((eq x 2) 3)
        ((eq x 3) 4)
        ((eq x 4) 5)
        ((eq x 5) 6)
        ((eq x 6) 7)
        ((eq x 7) 8)
        ((eq x 8) 9)
        ((eq x 9) :f)
        (t (bottom '(not a digit:) x))))
```

(pred symbol) → symbol | :f

```
(define (pred x)
  (cond ((eq x 1) 0)
        ((eq x 2) 1)
        ((eq x 3) 2)
        ((eq x 4) 3)
        ((eq x 5) 4)
        ((eq x 6) 5)
        ((eq x 7) 6)
```

```
((eq x 8) 7)
((eq x 9) 8)
((eq x 0) :f)
(t (bottom '(not a digit:) x)))
```

The **sum-of-digits** structure contains the sums of all combinations of digits in the form

```
(sum . carry)
```

The result of adding two digits *a* and *b* can be found in the *b*'th column of the *a*'th row of the structure. There are eleven columns in each row in order to support a single-bit carry value.

```
(define *sums-of-digits* '(
  ((0.0) (1.0) (2.0) (3.0) (4.0) (5.0) (6.0) (7.0) (8.0) (9.0) (0.1))
  ((1.0) (2.0) (3.0) (4.0) (5.0) (6.0) (7.0) (8.0) (9.0) (0.1) (1.1))
  ((2.0) (3.0) (4.0) (5.0) (6.0) (7.0) (8.0) (9.0) (0.1) (1.1) (2.1))
  ((3.0) (4.0) (5.0) (6.0) (7.0) (8.0) (9.0) (0.1) (1.1) (2.1) (3.1))
  ((4.0) (5.0) (6.0) (7.0) (8.0) (9.0) (0.1) (1.1) (2.1) (3.1) (4.1))
  ((5.0) (6.0) (7.0) (8.0) (9.0) (0.1) (1.1) (2.1) (3.1) (4.1) (5.1))
  ((6.0) (7.0) (8.0) (9.0) (0.1) (1.1) (2.1) (3.1) (4.1) (5.1) (6.1))
  ((7.0) (8.0) (9.0) (0.1) (1.1) (2.1) (3.1) (4.1) (5.1) (6.1) (7.1))
  ((8.0) (9.0) (0.1) (1.1) (2.1) (3.1) (4.1) (5.1) (6.1) (7.1) (8.1))
  ((9.0) (0.1) (1.1) (2.1) (3.1) (4.1) (5.1) (6.1) (7.1) (8.1) (9.1))
))
```

The **diffs-of-digits** structure contains the differences of all combinations of digits in the form

```
(difference . borrow)
```

The structure works in the same way as the above **sums-of-digits**. Because the “minus” operation is not commutative, it is important to look up the first operand to the difference operation in the rows and the second one in the columns of that row.

```
(define *diffs-of-digits* '(
  ((0.0) (9.1) (8.1) (7.1) (6.1) (5.1) (4.1) (3.1) (2.1) (1.1) (0.1))
  ((1.0) (0.0) (9.1) (8.1) (7.1) (6.1) (5.1) (4.1) (3.1) (2.1) (1.1))
  ((2.0) (1.0) (0.0) (9.1) (8.1) (7.1) (6.1) (5.1) (4.1) (3.1) (2.1))
  ((3.0) (2.0) (1.0) (0.0) (9.1) (8.1) (7.1) (6.1) (5.1) (4.1) (3.1))
  ((4.0) (3.0) (2.0) (1.0) (0.0) (9.1) (8.1) (7.1) (6.1) (5.1) (4.1))
  ((5.0) (4.0) (3.0) (2.0) (1.0) (0.0) (9.1) (8.1) (7.1) (6.1) (5.1))
  ((6.0) (5.0) (4.0) (3.0) (2.0) (1.0) (0.0) (9.1) (8.1) (7.1) (6.1))
  ((7.0) (6.0) (5.0) (4.0) (3.0) (2.0) (1.0) (0.0) (9.1) (8.1) (7.1))
  ((8.0) (7.0) (6.0) (5.0) (4.0) (3.0) (2.0) (1.0) (0.0) (9.1) (8.1))
  ((9.0) (8.0) (7.0) (6.0) (5.0) (4.0) (3.0) (2.0) (1.0) (0.0) (9.1))
))
```

The *%nth-item* function fetches the *d*'th item from the list *lst*. *D* must be a *digit* and not a zenlisp number. *Nth-item* is used to look up values in the above sum and difference tables.

zen style programming

(%nth-item digit list) → form

```
(define (%nth-item d lst)
  (cond ((eq d 0) (car lst))
        (t (%nth-item (pred d) (cdr lst)))))
```

%D+ adds two digits and a carry flag (represented by the digits 0 and 1) and delivers a pair consisting of their sum and a new carry value. It basically implements a decimal single-digit full adder.

(%d+ digit₁ digit₂ 0|1) → '(sum . carry)

```
(define (%d+ a b carry)
  (let ((row (%nth-item b *sums-of-digits*)))
    (cond ((eq carry 1) (%nth-item a (cdr row)))
          (t (%nth-item a row)))))
```

%D- subtracts a digit *b* and a borrow flag from a digit *a*. The borrow flag is represented by the digits 0 and 1. **%D-** and delivers a pair consisting of the difference and a new borrow flag.

(%d- digit₁ digit₂ 0|1) → '(difference . borrow)

```
(define (%d- a b carry)
  (let ((row (%nth-item a *diffs-of-digits*)))
    (cond ((eq carry 1) (%nth-item b (cdr row)))
          (t (%nth-item b row)))))
```

%D< is a predicate returning **:t** if the digit *a* has a smaller value than the digit *b*.

```
(define (%d< a b)
  (letrec
    ((dless
      (lambda (set)
        (cond ((null set)
              (bottom '(not digits:) a b))
              ((eq a (car set))
               (not (eq b (car set))))
              ((eq b (car set)) :f)
              (t (dless (cdr set)))))))
    (dless *digits*)))
```

Natural-p checks whether its argument is a natural number (a non-empty list of digits).

(natural-p form) → :t | :f

```
(define (natural-p x)
  (letrec
    ((lod-p
      (lambda (x)
        (cond ((null x) :t)
              ((atom x) :f)
              (t (and (digitp (car x))
                      (lod-p (cdr x)))))))
    (lod-p x)))
```

```
(and (not (atom x))
      (lod-p x)))
```

N-natural converts a number to a natural number. Because there are only natural number at this point, this is an identity operation.

(n-natural natural) → natural

```
(define n-natural id)
```

Normalize a natural number by removing leading zeroes.

(n-normalize natural) → natural

```
(define (n-normalize x)
  (cond ((null (cdr x)) x)
        ((eq (car x) 0)
         (n-normalize (cdr x)))
        (t x)))
```

Check whether two natural numbers are in strict ascending order. **N<** uses an empty **let** in order to close over %d<. Thi construct is used in packages to protect internal symbols from accidental redefinition.

(n< natural₁ natural₂) → :t | :f

```
(define n<
  (let ()
    (lambda (a b)
      (letrec
        ((d> (lambda (a b)
                (%d< b a)))
         (lt (lambda (a b r)
                (cond ((and (null a) (null b)) r)
                      ((null a) :t)
                      ((null b) :f)
                      (t (lt (cdr a)
                             (cdr b)
                             (cond ((%d< (car a) (car b)) :t)
                                   ((d> (car a) (car b)) :f)
                                   (t r)))))))
         (lt (reverse a) (reverse b) :f))))))
```

The other ordering predicates can be derived from **n<** easily:

(n> natural₁ natural₂) → :t | :f

(n<= natural₁ natural₂) → :t | :f

(n>= natural₁ natural₂) → :t | :f

```
(define (n> a b) (n< b a))
```

zen style programming

```
(define (n<= a b) (eq (n> a b) :f))
```

```
(define (n>= a b) (eq (n< a b) :f))
```

Check whether two natural numbers are equal.

(n= natural₁ natural₂) → :t | :f

```
(define (n= a b)
  (equal (n-normalize a)
        (n-normalize b)))
```

Add two natural numbers. The algorithm used here is basically the one that most people use when adding two numbers on a sheet of paper. It starts with the least significant digits and propagates a carry flag through the entire rows of digits. When one number has fewer digits than the other, the non-existent digits are substituted by 0.

(n+ natural₁ natural₂) → natural

```
(define n+
  (let ()
    (lambda (a b)
      (letrec
        ((add
          (lambda (a b c r)
            (cond ((null a)
                  (cond ((null b)
                        (cond ((eq c 0) r) ; no carry
                              (t (cons 1 r))))
                    (t (let ((sum (%d+ 0 (car b) c)))
                        (add ()
                            (cdr b)
                            (cdr sum)
                            (cons (car sum) r)))))))
              ((null b)
               (let ((sum (%d+ (car a) 0 c)))
                 (add (cdr a)
                     ()
                     (cdr sum)
                     (cons (car sum) r))))
              (t (let ((sum (%d+ (car a) (car b) c)))
                  (add (cdr a)
                      (cdr b)
                      (cdr sum)
                      (cons (car sum) r))))))))
    (add (reverse a) (reverse b) 0 ())))))
```

Subtract two natural numbers. Similar to **n+** above.

(n- natural₁ natural₂) → natural

```
(define n-
  (let ()
    (lambda (a b)
      (letrec
        ((diff
          (lambda (a b c r)
            (cond ((null a)
                  (cond
                     ((null b)
                      (cond ((eq c 0) r)
                            (t (bottom '(negative difference))))))
                     (t (bottom '(negative difference))))))
                 ((null b)
                  (cond ((eq c 0)
                        (append (reverse a) r))
                        (t (diff a '(1) 0 r))))))
          (t (let ((delta (%d- (car a) (car b) c)))
              (diff (cdr a)
                    (cdr b)
                    (cdr delta)
                    (cons (car delta) r))))))))
      (n-normalize (diff (reverse a) (reverse b) 0 ())))))
```

Test a natural number for being zero or one. These functions are equivalent to **(= x 0)** and **(= x 1)** respectively, but much more efficient.

(n-zero natural) → :t | :f

(n-one natural) → :t | :f

```
(define (n-zero x)
  (and (eq (car x) 0)
        (null (cdr x))))
```

```
(define (n-one x)
  (and (eq (car x) 1)
        (null (cdr x))))
```

Multiply two natural numbers. The algorithm uses decimal left shift operations to multiple by 10.

(n* natural₁ natural₂) → natural

```
(define (n* a b)
  (letrec
    ((*10
      (lambda (x)
        (append x '#0))))
    (add-n-times
      (lambda (a b r)

```


zenstyle programming

```
(cond ((n-zero (list b)) r)
      (t (add-n-times a (pred b) (n+ a r)))))
(times
 (lambda (a b r)
  (cond ((null b) r)
        (t (times (*10 a)
                   (cdr b)
                   (add-n-times a (car b) r))))))
(cond ((n-zero a) '#0)
      (t (times a (reverse b) '#0)))))
```

Divide two natural numbers, giving a list of the form (*quotient remainder*). The division algorithm works as follows:

- shift the divisor to the left until it has as many digits as the dividend;
- let **n** be the number of places by which the divisor was shifted;
- let the result **R** be '#0';
- do **n** times:
 - test how many times the divisor fits into the dividend; name this number **q**;
 - subtract **q** times the divisor from the dividend;
 - append **q** to **R**;²³
 - shift the divisor to the right by one digit.
- normalize the result **R**.

(n-divide natural₁ natural₂) → '*(quotient remainder)*

```
(define (n-divide a b)
  (letrec
    ; Equalize the divisor B by shifting it to the left
    ; (multiplying it by 10) until it has the same number
    ; of digits as the dividend A.
    ; Return: (new divisor . base 1 shift count)
    ((eq1
      (lambda (a b r s)
        (cond ((null a)
              (cons (reverse r) s))
              ((null b)
               (eq1 (cdr a)
                    ()
                    (cons 0 r)
                    (cons 'i s)))
              (t (eq1 (cdr a)
                      (cdr b)
                      (cons (car b) r)
                      s))))))
    ; Divide with quotient < 10
```

²³ Yes, zenlisp numbers can be appended using **append** because they are ordinary lists.

```
; Return (A/B*B . A/B)
(div10
  (lambda (a b r)
    (cond ((n< (car r) a)
           (div10 a b (cons (n+ (car r) b)
                             (n+ (cdr r) '#1))))
          ((equal (car r) a) r)
          (t (cons (n- (car r) b)
                    (n- (cdr r) '#1))))))

; X / 10
(dl0
  (lambda (x)
    (reverse (cdr (reverse x)))))
(div
  (lambda (a b r)
    (cond ((null (cdr b))
           (list (n-normalize r) a))
          (t (let ((quot (div10 a (car b) (cons '#0 '#0))))
                (div (n- a (car quot))
                      (cons (dl0 (car b)) (cddr b))
                      (append r (cdr quot))))))))
  (cond ((n-zero b) (bottom 'divide-by-zero))
        ((n< a b) (list '#0 a))
        (t (div a (eq1 a b) '#1) '#0))))
```

Divide two natural numbers and return only the quotient or the remainder.

(n-quotient natural₁ natural₂) → natural
(n-remainder natural₁ natural₂) → natural

```
(define (n-quotient a b) (car (n-divide a b)))
```

```
(define (n-remainder a b) (cadr (n-divide a b)))
```

Test a number for being even or odd. **Even** is basically **(n-zero (remainder x '#2))**, but more efficient.

(even natural) → natural
(odd natural) → natural

```
(define (even x)
  (and (memq (car (reverse x)) '#02468) :t))
```

```
(define (odd x) (eq (even x) :f))
```

Compute x raised to the y 'th power.

(n-expt natural₁ natural₂) → natural

```
(define (n-expt x y)
  (letrec
```

zen style programming

```
((square
  (lambda (x)
    (n* x x)))
 (n-expt1
  (lambda (y)
    (cond ((n-zero y) '#1)
          ((even y)
           (square (n-expt1 (n-quotient y '#2))))
          (t (n* x (square (n-expt1 (n-quotient y '#2))))))))
 (n-expt1 (n-natural y))))
```

Compute the greatest natural number that is not greater than the square root of the given argument. This function uses Newton's method.

(n-sqrt natural) → natural

```
(define (n-sqrt square)
  (letrec
    ((sqr
      (lambda (x last)
        (cond ((equal last x) x)
              ((equal last (n+ x '#1))
               (cond ((n> (n* x x) square) (n- x '#1))
                     (t x)))
              (t (sqr (n-quotient (n+ x (n-quotient square x))
                                   '#2)
                       x)))))
    (sqr square '#0)))
```

Compute the length of a list. This function is in the **nmath** package and not in **base**, because it uses numbers, which are not a trivial concept as you can see in this file.

(length list) → natural

```
(define (length x)
  (letrec
    ((len (lambda (x r)
            (cond ((null x) r)
                  (t (len (cdr x) (n+ r '#1))))))
    (len x '#0)))
```

Compute the *greatest common divisor* and *least common multiple* of two natural numbers.

(n-gcd natural) → natural

(n-lcm natural) → natural

```
(define (n-gcd a b)
  (cond ((n-zero b) a)
        ((n-zero a) b)
        ((n< a b) (n-gcd a (n-remainder b a)))
        (t (n-gcd b (n-remainder a b)))))
```

```
(define (n-lcm a b)
  (let ((cd (n-gcd a b)))
    (n* cd (n* (n-quotient a cd)
                (n-quotient b cd)))))
```

Find the limit **k** of a list of numbers **L** so that **k op x** for each **x** that is a member of **L** (without **k** if **op** imposes a strict order). **Op** must be a numeric predicate that imposes an order on **L**. When **op = <**, for example, *limit* returns the minimum of the list.

(limit function natural₁ natural₂ ...) → natural

```
(define (limit op a . b)
  (letrec
    ((lim (lambda (a)
             (cond ((null (cdr a)) (car a))
                   ((op (car a) (cadr a))
                    (lim (cons (car a) (cddr a))))
                   (t (lim (cdr a)))))))
    (lim (cons a b)))))
```

Find the maximum and minimum of a list using the *limit* function.

(max list) → natural

(min list) → natural

```
(define (n-max . a) (apply limit n> a))
```

```
(define (n-min . a) (apply limit n< a))
```

```
(require 'iter)
```

The following definitions specify the preferred names of the natural number operations. For example, ***** should be used in user-level code instead of **n***, because it is more readable, more flexible (because it is variadic), and because it does not depend on a specific math package.

The iterator functions from the **iter** package are used to make some of the binary **nmath** functions variadic. Only the **-** function is converted manually, because the natural “minus” operator does not make any sense with less than 2 arguments.

```
(define natural n-natural)

(define * (arithmetic-iterator n-natural n* '#1))

(define + (arithmetic-iterator n-natural n+ '#0))

(define (- . x)
  (cond ((or (null x) (null (cdr x)))
         (bottom '(too few arguments to n-natural -)))
        (t (fold (lambda (a b)
                    (n- (n-natural a) (n-natural b)))
```

zenstyle programming

```
(car x)
(cdr x))))))

(define < (predicate-iterator natural n<))

(define <= (predicate-iterator natural n<=))

(define = (predicate-iterator natural n=))

(define > (predicate-iterator natural n>))

(define >= (predicate-iterator natural n>=))

(define divide n-divide)

(define expt n-expt)

(define gcd (arithmetic-iterator natural n-gcd '#0))

(define lcm (arithmetic-iterator natural n-lcm '#1))

(define max n-max)

(define min n-min)

(define number-p natural-p)

(define one n-one)

(define quotient n-quotient)

(define remainder n-remainder)

(define sqrt n-sqrt)

(define zero n-zero)
```

13.4 integer math functions

The **imath** package extends the **nmath** package by adding support for negative numbers. Many of the functions defined here basically split integer numbers into signs and a natural numbers, rewrite the operation so that it can be carried out by the corresponding natural math function, compute the sign of the result and attach it to the intermediate result delivered by the **nmath** function.

The integer math package is contained in the file `imath.l`.

```
; zenlisp integer math functions
; By Nils M Holm, 2007, 2008
; Feel free to copy, share, and modify this code.
; See the file LICENSE for details.
```

```
; would use REQUIRE, but REQUIRE is in BASE
(cond ((defined 'base) :f)
      (t (load base)))
```

```
(define imath :t)
```

```
(require 'nmath)
```

Check whether a form represents an integer number. An integer number is either a natural number or a natural number prefixed with a plus (“+”) or minus (“-”) sign.

(integer-p form) → :t | :f

```
(define (integer-p a)
  (and (not (atom a))
       (or (natural-p a)
           (and (memq (car a) '#+ -)
                (natural-p (cdr a))))))
```

Convert an integer or natural number to an integer.

(i-integer natural | integer) → integer

```
(define (i-integer a)
  (cond ((eq (car a) '+) (cdr a))
        ((eq (car a) '-') a)
        ((digitp (car a)) a)
        (t (bottom (list 'i-integer a)))))
```

Convert a positive integer or natural number to a natural number.

(i-natural natural | integer) → natural

```
(define (i-natural a)
  (cond ((eq (car a) '+) (cdr a))
        ((digitp (car a)) a)
        (t (bottom (list 'i-natural a)))))
```

Normalize an integer number by removing leading zeroes and a positive sign.

(i-normalize integer) → integer

```
(define (i-normalize x)
  (cond ((eq (car x) '+)
        (n-normalize (cdr x)))
        ((eq (car x) '-')
         (let ((d (n-normalize (cdr x))))
           (cond ((n-zero d) d)
                 (t (cons '- d)))))
        (t (n-normalize x))))
```

zen style programming

Check whether the given integer is negative. This function is equivalent to `(< x 0)`, but more efficient.

(i-negative integer) → :t | :f

```
(define (i-negative x) (eq (car x) '-))
```

Remove the sign of an integer, thereby returning its absolute value.

(i-abs integer) → natural

```
(define (i-abs x)
  (cond ((i-negative x) (cdr x))
        ((eq (car x) '+) (cdr x))
        (t x)))
```

Check whether a given integer is equal to zero or one. Like their natural counterparts these functions are performance hacks.

(i-zero integer) → :t | :f

(i-one integer) → :t | :f

```
(define (i-zero x)
  (n-zero (i-abs x)))
```

```
(define (i-one x)
  (and (n-one (i-abs x))
       (neq (car x) '-)))
```

Negate an integer. This function is equivalent to `(- 0 x)`, but more efficient.

(i-negate integer) → integer

```
(define (i-negate x)
  (cond ((n-zero (i-abs x)) x)
        ((eq (car x) '-') (cdr x))
        ((eq (car x) '+) (cons '- (cdr x)))
        (t (cons '- x))))
```

Add two integer numbers. This function handles only the signs and uses **n+** and **n-** to compute actual sums. In order to be able to do all computations using natural numbers, it rewrites its operations as outlined in the following table:²⁴

Original term	Rewritten term
<code>+a + +b</code>	<code>a + b</code>
<code>+a + -b</code>	<code>a - b </code> if <code> a > b </code>
<code>+a + -b</code>	<code>-(b - a)</code> if <code> a <= b </code>

²⁴ `|x|` denotes the absolute value of `x`.

```
-a + +b      -( |a| - b)      if |a| >  |b|
-a + +b      b - |a|          if |a| <= |b|
-a + -b      -( |a| + |b| )
```

(i+ integer₁ integer₂) → integer

```
(define (i+ a b)
  (cond ((and (not (i-negative a))
              (not (i-negative b)))
         (n+ (i-abs a) (i-abs b)))
        ((and (not (i-negative a))
              (i-negative b))
         (cond ((n> (i-abs a) (i-abs b))
                (n- (natural a) (i-abs b)))
               (t (i-negate (n- (i-abs b) (natural a))))))
        ((and (i-negative a)
              (not (i-negative b)))
         (cond ((n> (i-abs a) (i-abs b))
                (i-negate (n- (i-abs a) (natural b))))
               (t (n- (natural b) (i-abs a)))))
        (t (i-negate (n+ (i-abs a) (i-abs b))))))
```

Subtract two integer numbers. This function handles only the signs and delegates the computations to **n-** and **i+**. It rewrites its operations as follows:

Original term	Rewritten term
+a - +b	-(b - a) if a < b
+a - +b	a - b if a >= b
+a - -b	a + b
-a - +b	a + -b
-a - -b	a + b

(i- integer₁ integer₂) → integer

```
(define (i- a b)
  (cond ((i-negative b)
         (i+ a (i-abs b)))
        ((i-negative a)
         (i+ a (i-negate b)))
        ((n< (i-abs a) (i-abs b))
         (i-negate (n- (i-abs b) (i-abs a))))
        (t (n- (i-abs a) (i-abs b)))))
```

Check whether two integer numbers are in strict ascending order. This function first checks the signs to compare the numbers and delegates its task to **n<** only if the signs are equal.

zen style programming

(i< integer₁ integer₂) → :t | :f

```
(define (i< a b)
  (cond ((i-negative a)
        (cond ((not (i-negative b)) :t)
              (t (n< (i-abs b) (i-abs a)))))
        ((i-negative b) :f)
        (t (n< (i-abs a) (i-abs b)))))
```

As usual, the remaining ordering predicates can be derived from the “less than” relation.

(i> integer₁ integer₂) → :t | :f
(i<= integer₁ integer₂) → :t | :f
(i>= integer₁ integer₂) → :t | :f

```
(define (i> a b) (i< b a))

(define (i<= a b) (eq (i> b a) :f))

(define (i>= a b) (eq (i< b a) :f))
```

Check whether two integers are equal.

(i= integer₁ integer₂) → :t | :f

```
(define (i= a b)
  (equal (i-normalize a)
         (i-normalize b)))
```

Multiply two integers. Handle only signs, delegate the rest to **n***.

(i* integer₁ integer₂) → integer

```
(define (i* a b)
  (cond ((zero a) '#0)
        ((eq (i-negative a) (i-negative b))
         (n* (i-abs a) (i-abs b)))
        (t (i-negate (n* (i-abs a) (i-abs b))))))
```

Divide two integers. Handle only signs, delegate the rest to **n-divide**. Like **n-divide**, this function returns both the quotient and the remainder in a list.

(i-divide integer₁ integer₂) → '(quotient remainder)

```
(define (i-divide a b)
  (letrec
    ((sign
      (lambda (x)
        (cond ((eq (i-negative a) (i-negative b)) x)
              (t (cons '- x)))))
     (rsign
```

```
(lambda (x)
  (cond ((i-negative a) (cons '- x))
        (t x))))
(idiv
 (lambda (a b)
  (cond ((n-zero b) (bottom '(divide by zero)))
        ((n< (i-abs a) (i-abs b))
         (list '#0 (rsign (i-abs a))))
        (t (let ((q (n-divide (i-abs a) (i-abs b))))
              (list (sign (car q))
                    (rsign (cadr q))))))))
 (idiv (i-integer a) (i-integer b))))
```

Divide two integers and return only the quotient or the remainder.

(i-quotient integer₁ integer₂) → integer
(i-remainder integer₁ integer₂) → integer

```
(define (i-quotient a b) (car (i-divide a b)))
```

```
(define (i-remainder a b) (cadr (i-divide a b)))
```

Compute the modulus of two integers. Because the modulus is specific to integers, there is no version with an “i” prefix.

(modulo integer₁ integer₂) → integer

```
(define (modulo a b)
  (let ((rem (i-remainder a b)))
    (cond ((i-zero rem) '#0)
          ((eq (i-negative a)
               (i-negative b))
           rem)
          (t (i+ b rem)))))
```

Compute x raised to the power of y . Handle only signs, delegate the rest to **n-expt**. I guess you got the idea by now...

(i-expt integer₁ integer₂) → integer

```
(define (i-expt x y)
  (letrec
    ((i-expt2
      (lambda (x y)
        (cond ((or (not (i-negative x))
                   (even y))
              (n-expt (i-abs x) y))
              (t (i-negate (n-expt (i-abs x) y))))))
    (i-expt2 (i-integer x) (natural y))))
```

zen style programming

Define integer maximum and minimum in terms of **limit**.

```
(i-max integer1 integer2 ...) → integer  
(i-min integer1 integer2 ...) → integer
```

```
(define (i-max . a) (apply limit i> a))
```

```
(define (i-min . a) (apply limit i< a))
```

I-sqrt is similar to **n-sqrt**, but rejects negative operands.

```
(i-sqrt integer) → integer
```

```
(define (i-sqrt x)  
  (cond ((i-negative x)  
        (bottom (list 'i-sqrt x)))  
        (t (n-sqrt x))))
```

The integer versions of **gcd** and **lcm** just cut off the signs.

```
(i-gcd integer ...) → integer  
(i-lcm integer ...) → integer
```

```
(define (i-gcd a b)  
  (n-gcd (i-abs a) (i-abs b)))
```

```
(define (i-lcm a b)  
  (n-lcm (i-abs a) (i-abs b)))
```

```
(require 'iter)
```

As in the **nmath** package, the remainder of the file defines the preferred names of the math functions. Many functions defined in **nmath** get redefined here and some new ones are added.

The only notable modification is the extension of the “minus” operator. When the **-** function of **imath** is applied to a single argument, it negates that argument.

```
(define integer i-integer)  
  
(define * (arithmetic-iterator integer i* '#1))  
  
(define + (arithmetic-iterator integer i+ '#0))  
  
(define (- . x)  
  (cond ((null x)  
        (bottom '(too few arguments to integer -)))  
        ((eq (cdr x) ())  
         (i-negate (car x)))  
        (t (fold (lambda (a b)  
                   (i- (integer a) (integer b)))  
                  (car x)  
                  (cdr x))))))
```

```
(define < (predicate-iterator integer i<))

(define <= (predicate-iterator integer i<=))

(define = (predicate-iterator integer i=))

(define > (predicate-iterator integer i>))

(define >= (predicate-iterator integer i>=))

(define abs i-abs)

(define divide i-divide)

(define expt i-expt)

(define gcd (arithmetic-iterator integer i-gcd '#0))

(define lcm (arithmetic-iterator integer i-lcm '#1))

(define max i-max)

(define min i-min)

(define natural i-natural)

(define negate i-negate)

(define negative i-negative)

(define number-p integer-p)

(define one i-one)

(define quotient i-quotient)

(define remainder i-remainder)

(define sqrt i-sqrt)

(define zero i-zero)
```

13.5 rational math functions

The **rmath** package extends the **imath** package by adding support for rational numbers. The functions defined here basically split rational numbers into numerators and denominators, rewrite the operations so that they can be done by the integer math functions and return a result that has the least applicable type.

zen style programming

This means that rational number operations may return a value that has a lesser type than “rational” if that type is sufficient to represent the result. For example:

```
(+ '#1/3 '#2/3) => '#1
```

Because the sum of 1/3 and 2/3 can be represented by a natural number, the rational + operator returns one.

The rational math package is contained in the file `rmath.l`.

```
; zenlisp rational math functions
; By Nils M Holm, 2007, 2008
; Feel free to copy, share, and modify this code.
; See the file LICENSE for details.

; would use REQUIRE, but REQUIRE is in BASE
(cond ((defined 'base) :f)
      (t (load base)))

(define rmath :t)

(require 'imath)
```

Extract the numerator and denominator of a rational number.

(numerator rational) → integer
(denominator rational) → integer

```
(define (numerator x)
  (reverse (cdr (memq '/ (reverse x)))))

(define (denominator x) (cdr (memq '/ x)))
```

Check whether a form represents a rational number.

(rational-p form) → :t | :f

```
(define (rational-p x)
  (and (listp x)
        (memq '/ x)
        (integer-p (numerator x))
        (integer-p (denominator x))))
```

Check whether a form represents a number (either rational or integer or natural).

(r-number-p form) → :t | :f

```
(define (r-number-p x)
  (or (integer-p x)
      (rational-p x)))
```

Create a rational number from a given numerator and denominator.

(make-rational integer₁ integer₂) → rational

```
(define (make-rational num den)
  (append num '#/ den))
```

Convert any type of number to a rational number. When the number is not already rational, add a denominator of '#1. Note again that numeric types are distinguished by *syntax*, so '#5 is not a rational number, but '#5/1 is one.

(rational number) → rational

```
(define (rational x)
  (cond ((rational-p x) x)
        (t (make-rational x '#1))))
```

Rational versions of the **zero** and **one** functions.

(r-zero number) → :t | :f

(r-one number) → :t | :f

```
(define (r-zero x)
  (cond ((rational-p x) (r= x '#0))
        (t (i-zero x))))
```

```
(define (r-one x)
  (cond ((rational-p x) (r= x '#1))
        (t (i-one x))))
```

Reduce a rational number to its least terms.

(%least-terms rational) → rational

```
(define (%least-terms x)
  (let ((cd (gcd (numerator x) (denominator x))))
    (cond ((r-one cd) x)
          (t (make-rational (quotient (numerator x) cd)
                             (quotient (denominator x) cd))))))
```

Convert rationals with denominators of '#1 to a lesser type.

(%decay rational) → rational | integer

```
(define (%decay x)
  (cond ((r-one (denominator x))
        (numerator x))
        (t x)))
```

Normalize a rational number. This is explained in depth on pages 52f. The empty **let** closes over *%least-terms* and *%decay*.

zen style programming

(r-normalize number) → rational | integer

```
(define r-normalize
  (let ()
    (lambda (x)
      (letrec
        ((norm-sign (lambda (x)
                      (let ((num (numerator x))
                          (den (denominator x)))
                        (let ((pos (eq (i-negative num)
                                      (i-negative den))))
                          (make-rational (cond (pos (i-abs num))
                                              (t (cons '- (i-abs num))))
                                           (i-abs den))))))
          (cond ((rational-p x)
                 (%decay (%least-terms (norm-sign x))))
                (t (i-normalize x)))))))
```

Convert a rational or integer number to type integer/natural. Unlike their integer counterparts (**i-integer** and **i-natural**), these versions also accept rationals that can be reduced to integers/naturals.

(r-integer number) → integer

(r-natural number) → natural

```
(define (r-integer x)
  (let ((xlt (+ '#0 x)))
    (cond ((rational-p xlt)
          (bottom (list 'r-integer x)))
          (t xlt))))

(define (r-natural x)
  (i-natural (r-integer x)))
```

Compute the absolute value of a rational number.

(r-abs number) → rational | integer

```
(define (r-abs x)
  (cond ((rational-p x)
        (make-rational (i-abs (numerator x))
                       (i-abs (denominator x))))
        (t (i-abs x))))
```

Equalize the denominators of two rational numbers, so they can be added, subtracted or compared. Quick: what is greater, '#123/456 or '#213/789? *Equalize* gives the answer:

```
(%equalize '#123/456 '#213/789) => '(#32349/119928 #32376/119928)
```

However, there is no need to use *%equalize* in user-level code, because it is used by the rational < predicate, which is introduced later in this section.

(%equalize rational₁ rational₂) → list

```
(define (%equalize a b)
  (let ((num-a (numerator a))
        (num-b (numerator b))
        (den-a (denominator a))
        (den-b (denominator b)))
    (let ((cd (gcd den-a den-b)))
      (cond
        ((r-one cd)
         (list (make-rational (i* num-a den-b)
                              (i* den-a den-b))
               (make-rational (i* num-b den-a)
                              (i* den-b den-a)))))
        (t (list (make-rational (quotient (i* num-a den-b) cd)
                                   (quotient (i* den-a den-b) cd))
                  (make-rational (quotient (i* num-b den-a) cd)
                                   (quotient (i* den-b den-a) cd)))))))
```

You already know this principle from the **imath** package: **R+**, **r-**, and **r*** handle only things that are specific to operations on rational numbers and delegate the rest to their integer counterparts.

(r+ number₁ number₂) → rational | integer

(r- number₁ number₂) → rational | integer

(r* number₁ number₂) → rational | integer

```
(define r+
  (let ()
    (lambda (a b)
      (let ((factors (%equalize (rational a) (rational b)))
            (radd
              (lambda (a b)
                (r-normalize
                 (make-rational (i+ (numerator a) (numerator b))
                               (denominator a))))))
        (radd (car factors) (cadr factors))))))

(define r-
  (let ()
    (lambda (a b)
      (let ((factors (%equalize (rational a) (rational b)))
            (rsub
              (lambda (a b)
                (r-normalize
                 (make-rational (i- (numerator a) (numerator b))
                               (denominator a))))))
        (rsub (car factors) (cadr factors))))))

(define (r* a b)
  (let ((rmul
        (lambda (a b)
```


zen style programming

```
(r-normalize
  (make-rational (i* (numerator a) (numerator b))
                 (i* (denominator a) (denominator b))))))
(rmul (rational a) (rational b))))
```

There is no integer version of $/$, but because

$$(a/b) / (c/d) = (a*d) / (b*c)$$

rational division is easily delegated to **i***.

(r/ number₁ number₂) → rational | integer

```
(define (r/ a b)
  (let ((rdiv
        (lambda (a b)
          (r-normalize
            (make-rational (i* (numerator a) (denominator b))
                           (i* (denominator a) (numerator b)))))))
    (cond ((r-zero b) (bottom (list 'r/ a b)))
          (t (rdiv (rational a) (rational b))))))
```

R< uses *%equalize* and **i<** to compare rational numbers. The other relational predicates are derived from it. You already know this procedure from the other math packages .

(r< number₁ number₂) → :t | :f
(r> number₁ number₂) → :t | :f
(r<= number₁ number₂) → :t | :f
(r>= number₁ number₂) → :t | :f

```
(define r<
  (let ()
    (lambda (a b)
      (let ((factors (%equalize (rational a) (rational b))))
        (i< (numerator (car factors))
             (numerator (cadr factors)))))))

(define (r> a b) (r< b a))

(define (r<= a b) (eq (r> a b) :f))

(define (r>= a b) (eq (r< a b) :f))
```

Check whether two numbers (no matter which type they have) are equal.

(r= number₁ number₂) → :t | :f

```
(define r=
  (let ()
    (lambda (a b)
      (cond ((or (rational-p a) (rational-p b))
```

```
(equal (%least-terms (rational a))
      (%least-terms (rational b))))
(t (i= a b))))))
```

Raise x to the y 'th power. Unlike its integer counterpart this function accepts negative exponents.

(r-expt number₁ number₂) → rational | integer

```
(define (r-expt x y)
  (letrec
    ((rx (cond ((i-negative (r-integer y))
                (r/ '#1 (rational x)))
               (t (rational x))))
     (square
      (lambda (x)
        (r* x x)))
     (exp
      (lambda (x y)
        (cond ((r-zero y) '#1)
              ((even y)
               (square (exp x (quotient y '#2))))
              (t (r* x (square (exp x (quotient y '#2))))))))
    (exp rx (i-abs (r-integer y)))))
```

Check whether a number is negative.

(r-negative number) → :t | :f

```
(define (r-negative x)
  (cond ((rational-p x)
        (i-negative (numerator (r-normalize x))))
        (t (i-negative x))))
```

Negate a number.

(r-negate number) → rational | integer

```
(define (r-negate x)
  (cond ((rational-p x)
        (let ((nx (r-normalize x)))
          (make-rational (i-negate (numerator nx))
                        (denominator nx))))
        (t (i-negate x))))
```

Define the rational maximum and minimum functions in terms of **limit**.

(r-max number₁ number₂ ...) → rational | integer

(r-min number₁ number₂ ...) → rational | integer

```
(define (r-max . a) (apply limit r> a))
```

```
(define (r-min . a) (apply limit r< a))
```

zenstyle programming

Compute the square root of a number. The *precision* argument specifies the maximum error of the result. A value of 10, for example, means that the value returned by this function may not differ from the actual square root of the argument by a value that is greater than 10^{-10} (or 0.0000000001).

(r-sqrt number natural) → rational | integer

```
(define (r-sqrt square precision)
  (let ((e (make-rational '#1 (r-expt '#10 (r-natural precision)))))
    (letrec
      ((sqr (lambda (x)
               (cond ((r< (r-abs (r- (r* x x) square))
                          e)
                     x)
                     (t (sqr (r/ (r+ x (r/ square x))
                                '#2)))))))
      (sqr (n-sqrt (r-natural square))))))

(require 'iter)
```

As in the other math packages, the remainder of the file defines the preferred names of the functions. The `-` and `/` functions accept at least one argument. When `/` is applied to a single argument, it returns its reciprocal value ($1/x$).

```
(define * (arithmetic-iterator rational r* '#1))

(define + (arithmetic-iterator rational r+ '#0))

(define (- . x)
  (cond ((null x)
        (bottom '(too few arguments to rational -)))
        ((eq (cdr x) ()) (r-negate (car x)))
        (t (fold (lambda (a b)
                    (r- (rational a) (rational b)))
                  (car x)
                  (cdr x)))))

(define (/ . x)
  (cond ((null x)
        (bottom '(too few arguments to rational /)))
        ((eq (cdr x) ())
         (/ '#1 (car x)))
        (t (fold (lambda (a b)
                    (r/ (rational a) (rational b)))
                  (car x)
                  (cdr x)))))

(define < (predicate-iterator rational r<))

(define <= (predicate-iterator rational r<=))
```

300

```
(define = (predicate-iterator rational r=))  
(define > (predicate-iterator rational r>))  
(define >= (predicate-iterator rational r>=))  
(define abs r-abs)  
(define *epsilon* '#10)  
(define expt r-expt)  
(define integer r-integer)  
(define max r-max)  
(define min r-min)  
(define natural r-natural)  
(define negate r-negate)  
(define negative r-negative)  
(define number-p r-number-p)  
(define one r-one)  
(define (sqrt x) (r-sqrt x *epsilon*))  
(define zero r-zero)
```

zen style programming

appendix

A.1 tail call rules

A *tail-recursive* program is a program that recurses by using tail calls exclusively.

A tail call is a function application that is in a tail position.

This is an exhaustive list of tail positions in zenlisp:

(lambda (...) (function ...))

The outermost function application in a function body.

(let (...) body)

The bodies of **let**.

(letrec (...) body)

The bodies of **letrec**.

(apply function ...)

Applications of **apply**.

(and ... expression)

The *last* argument of **and**.

(or ... expression)

The *last* argument of **or**.

(cond ... (predicate body) ...)

Each body of **cond**.

Note that tail call rules may be combined. The application of **f** in the following example is in a tail position:

```
(lambda ()  
  (let ()  
    (cond (t (or :f :f (f))))))
```

A.2 zenlisp functions

The following symbols are used in this summary:

Symbol	Meaning
<code>alist</code>	an association list
<code>expr</code>	any type of expression
<code>form</code>	any type of form (unevaluated)
<code>fun</code>	a function or closure
<code>name</code>	a symbol (unevaluated)
<code>pair</code>	a pair
<code>symbol</code>	a symbol
<code>a b</code>	either <i>a</i> or <i>b</i>
<code>a...</code>	zero, one, or multiple instances of <i>a</i>

A.2.1 definitions

(define name expr)

Define constant *name* with value *expr*.

(define (name₁ name₂ ...) expr)

Define function *name₁* with optional variables *name₂*... and body *expr*.

(defined symbol)

Test whether *symbol* is defined.

(lambda (name ...) expr) | (lambda name expr)

Create closure with variables *name*... or *name* and body *expr*.

(let ((name₁ expr₁) ...) expr)

Create an environment with the bindings *name_i=expr_i*... and evaluate *expr* in that environment.

(letrec ((name₁ expr₁) ...) expr)

Create an environment with the recursive bindings *name_i=expr_i*... and evaluate *expr* in that environment.

(quote form)

Create a datum.

(recursive-bind alist)

Fix recursive bindings in environments.

A.2.2 control

(and expr ...)

Reduce expressions. Return the first one giving :**f** or the last one.

(apply fun expr ... list)

Apply *fun* to the optional expressions and the members of *list*.

(bottom expr ...)

Reduce to an undefined value.

(cond (expr_p expr) ...)

Reduce to the first *expr* whose associated *expr_p* evaluates to truth.

(eval expr)

Reduce *expr* to its normal form.

(or expr ...)

Reduce expressions. Return the first one evaluating to truth or the last one.

A.2.3 lists

(append list ...)

Append lists.

(assoc expr alist)

Find association with key=*expr* in association list *alist*; else return :**f**.

(assq symbol alist)

Find association with key=*symbol* in association list *alist*; else return :**f**.

(caar pair) ... (cddddr pair)

Extract parts of nested pairs. **Caar** = car of car, **cadr** = car of cdr, etc.

(car pair)

Extract car part of *pair*.

(cdr pair)

Extract cdr part of *pair*.

(cons expr₁ expr₂)

Construct fresh pair ' (*expr₁* . *expr₂*).

(equal expr₁ expr₂)

Test whether *expr₁* is equal to (looks the same as) *expr₂*.

zen style programming

(explode symbol)

Decompose a symbol into a list of single-character symbols.

(fold fun expr list)

Fold *fun* over *list* with base value *expr*. Left-associative version.

(fold-r fun expr list)

Fold *fun* over *list* with base value *expr*. Right-associative version.

(implode list)

Compose a symbol from a list of single-character symbols.

(list expr ...)

Create a list with the given members.

(listp expr)

Test whether *expr* is a proper (non-dotted) list.

(map fun list₁ list₂ ...)

Map function *fun* over the given lists.

(member expr list)

Find the first sublist of *list* starting with *expr*, else return :f.

(memq symbol list)

Find the first sublist of *list* starting with *symbol*, else return :f.

(null expr)

Test whether *expr* is ().

(reverse list)

Return a reverse copy of *list*.

A.2.4 miscellanea

(atom expr)

Test whether *expr* is atomic (either a symbol or ()).

(eq expr₁ expr₂)

Test whether *expr₁* and *expr₂* are identical.

(id expr)

Identity function (return *expr*).

(neq expr₁ expr₂)

Test whether *expr₁* and *expr₂* are *not* identical.

(not *expr*)

Test whether *expr* is identical to :**f** (logical “not”).

A.2.5 packages

(require *symbol*)

Load the given package if not already in memory.

A.2.6 meta functions

(OK, this is not really a function.) This variable is always bound to the latest toplevel result, i.e. the normal form that was most recently printed by the interpreter.

(closure-form *args* | *body* | *env*)

Control how much of a closure is printed.

(dump-image *name*)

Dump workspace image to file *name*. Reload the image by passing *name* to zenlisp.

(gc)

Run garbage collection and return statistics.

(load *name*)

Load definitions from the file *name*.

(quit)

End a zenlisp session.

(stats *expr*)

Reduce *expr* to normal form and return some statistics.

(symbols)

Return a list of all symbols in the symbol table.

(trace *name*) | **(trace)**

Trace the function with the given *name*. **(Trace)** switches tracing off.

(verify-arrows :t | :f)

Turn verification of => operators on or off.

A.3 math functions

Symbols used in this summary:

Symbol	Meaning
x	any number
r	rational number
i	integer number
n	natural number
$[x]$	x is optional
$a b$	either a or b
$a\dots$	zero, one, or multiple instances of a

Function	Returns...
<code>(* x ...)</code> $\Rightarrow x$	product
<code>*epsilon*</code> $\Rightarrow n$	\log_{10} of precision of <code>sqrt</code>
<code>(+ x ...)</code> $\Rightarrow x$	sum
<code>(- x1 x2 x3 ...)</code> $\Rightarrow x$	difference
<code>(- x)</code> $\Rightarrow x$	negative number
<code>(/ x1 x2 x3 ...)</code> $\Rightarrow x$	ratio
<code>(< x1 x2 x3 ...)</code> $\Rightarrow :t :f$	$:t$ for strict ascending order
<code>(<= x1 x2 x3 ...)</code> $\Rightarrow :t :f$	$:t$ for strict non-descending order
<code>(= x1 x2 x3 ...)</code> $\Rightarrow :t :f$	$:t$ for equivalence
<code>(> x1 x2 x3 ...)</code> $\Rightarrow :t :f$	$:t$ for strict descending order
<code>(>= x1 x2 x3 ...)</code> $\Rightarrow :t :f$	$:t$ for strict non-ascending order
<code>(abs x)</code> $\Rightarrow x$	absolute value
<code>(denominator r)</code> $\Rightarrow i$	denominator
<code>(divide i1 i2)</code> $\Rightarrow '(i3 i4)$	quotient $i3$ and remainder $i4$
<code>(even i)</code> $\Rightarrow :t :f$	$:t$, if i is even
<code>(expt x i)</code> $\Rightarrow x$	x to the power of i
<code>(gcd i1 i2 ...)</code> $\Rightarrow n$	greatest common divisor
<code>(integer x)</code> $\Rightarrow i$	an integer with the value x
<code>(integer-p x)</code> $\Rightarrow :t :f$	$:t$, if x is integer
<code>(lcm i1 i2 ...)</code> $\Rightarrow n$	least common multiple
<code>(length list)</code> $\Rightarrow n$	length of a list
<code>(max x1 x2 ...)</code> $\Rightarrow x$	maximum value
<code>(min x1 x2 ...)</code> $\Rightarrow x$	minimum value
<code>(modulo i1 i2)</code> $\Rightarrow i3$	modulus

<code>(natural x) => n</code>	a natural with the value x
<code>(natural-p x) => :t :f</code>	:t, if x is natural
<code>(negate i r) => i r</code>	negative value
<code>(negative x) => :t :f</code>	:t, if x is negative
<code>(number-p expr) => :t :f</code>	:t, if <i>expr</i> represents a number
<code>(numerator r) => i</code>	numerator
<code>(odd i) => :t :f</code>	:t, if i is not even
<code>(one x) => :t :f</code>	:t, if x equals one
<code>(quotient i1 i2) => i</code>	quotient
<code>(rational x) => r</code>	a rational with the value x
<code>(rational-p x) => :t :f</code>	:t, if x is rational
<code>(remainder i1 i2) => i</code>	division remainder
<code>(sqrt n) => x</code>	square root, see also *epsilon*
<code>(zero x) => :t :f</code>	:t, if x equals zero

A.4 working with zenlisp

Meta functions alter the state of the zenlisp system.
--

The **load** meta function reads a text file and reduces all expressions contained in that file to their normal forms.

Given a file named `palindrome.l` containing the lines

```
(define (palindrome x)
  (append x (reverse x)))
```

the function application

```
(load palindrome)
```

will load the above definition.

Load automatically appends the `.l` suffix to the file name.

Loading the same file again will update all definitions of that file.

When the file name begins with a tilde, **load** loads a zenlisp package from a pre-defined location. For instance

```
(load ~nmath)
```

loads the natural math functions into the zenlisp system.

zen style programming

The actual location of the system packages is specified by the `$ZENSRC` environment variable.

While **load** is typically used to load packages interactively, **require** is used to make a program dependent on a *package* [page 62].

A program beginning with the function application

```
(require '~rmath)
```

depends on the **rmath** package.

Because **require** is a lambda function (and not an internal pseudo function), its argument has to be quoted. Unlike **load**, **require** never loads a package twice:

```
(require '~rmath) => :t  
(require '~rmath) => :f
```

Hence it can be used to load mutually dependent packages.

The **dump-image** meta function dumps the complete `zenlisp` workspace to a file:

```
(load ~rmath)  
(load ~amk)  
(dump-image my-workspace)
```

This session creates a new image file named `my-workspace` which contains the **rmath** and **amk** packages.

To load an image, pass the image file name to `zenlisp` (% is the prompt of the Unix shell):

```
% zl my-workspace
```

The **trace** meta function makes the interpreter trace applications of a specific function:

```
(define (d x) (or (atom x) (d (cdr x))))  
(d '#xyz) => :t  
(trace d) => :t  
(d '#xyz)  
+ (d #xyz)  
+ (d #yz)  
+ (d #z)  
+ (d ())  
=> :t
```

Applying **trace** to no arguments switches tracing off:

```
(trace) => :t
```

The **stats** meta function measures the resources used during the reduction of an expression:

```
(stats (append '#abc '#def))  
=> '(#abcdef #240 #1,213 #0)
```

Its normal form is a list containing the following information:

```
'(normal-form steps nodes gcs)
```

where

- *normal-form* is the normal form of the expression to evaluate;
- *steps* is the number of reduction steps performed;
- *nodes* is the number of nodes allocated during reduction;
- *gcs* is the number of garbage collections performed during reduction.

The **verify-arrows** function switches verification mode on or off. Passing **:t** to it enables verification, **:f** disables it:

```
(verify-arrows t) => :t
```

In verification mode, **=>** operators are verified by making sure that the normal form on the lefthand side of each **=>** matches its righthand side:

```
(cons 'heads 'tails) => '(heads . tails) ; OK  
(cons 'heads 'tails) => 'foo ; FAIL
```

As long as an expression reduces to the expected normal form, nothing special happens. When the forms do not match, though, an error is reported by **zenlisp**:

```
(cons 'heads 'tails) => 'foo  
=> '(heads . tails)  
* 2: REPL: Verification failed; expected: 'foo
```

In non-verifying mode **=>** introduces a comment to the end of the line (like **;**), thereby facilitating the cutting and pasting of expressions.

The **quit** meta function ends a **zenlisp** session:

```
(quit)
```

For obvious reasons, **(quit)** has no normal form.

A.4.1 the development cycle

Although you can enter whole programs at the read-eval-print loop (REPL), doing so might turn out to be a bit inconvenient, because the **zenlisp** REPL lacks all but the most rudimentary editing features.

So it is recommended that you use a text editor of your choice to enter or modify **zenlisp** programs. For instance, you may have typed the following code and saved it to the file **hanoi.l**:²⁵

```
(require '~nmath)
```

²⁵ **Zenlisp** source code *must* have a **.l** suffix or the interpreter cannot load it.

zen style programming

```
(define (hanoi n)
  (letrec
    ((h (lambda (n from to via)
          (cond ((zero n) ())
                (t (append (h (- n '#1) from via to)
                           (list (liat from to))
                           (h (- n '#1) via to from)))))))
    (h n 'from 'to 'via)))
```

The most practical approach to test the program is to keep a window or virtual terminal open that runs a `zenlisp` process. After saving the above program, go to the `zenlisp` terminal and load the program (user input is in *italics*):

```
(load hanoi)
* hanoi.l: 10: REPL: wrong argument count: (define (hanoi n) (letrec ((h
(lambda (n from to via) (cond ((zero n) ()) (t (append (h (- n '#1) from
via to) (list (liat from to)) (h (- n '#1) via to from)))))) (h n 'from
'to 'via)))
```

The interpreter detects some syntax errors already at load time, like the above one. It informs you that line 10 of the file `hanoi.l` contained a function application with a wrong number of arguments. The error occurred at the top level (REPL).

At this point, an editor that can highlight or otherwise match parentheses is a great help. Using it, you will quickly discover that the **letrec** form of the file ends prematurely in the following line:

```
(h (- n '#1) via to from))))))
```

Deleting the superfluous closing parenthesis should fix the problem and indeed, when you reload the program in the interpreter window, it works fine (so far):

```
(load hanoi)
=> :t
```

The next step is to feed some input to the program:

```
(hanoi '#3)
* 4: h: symbol not bound: liat
* Trace: h h
```

Each line printed by the interpreter that begins with an asterisk indicates trouble. In the above case it is caused by an unbound symbol named *liat*. The line number is useless in this case, because it refers to the REPL and not to a file. However, we can see that the error occurred in the *h* function. Indeed this function contains a misspelled instance of **list**, so this error is a simple typo. Just return to the editor session and fix this bug, too:

```
(list (list from to))
```

Then go back to the interpreter and load the code again. This time, it should work fine:

```
(load hanoi)
=> :t
(hanoi '#3)
=> '((from to) (from via) (to via) (from to) (via from) (via to) (from to))
```

Great. BTW: this program is the native `zenlisp` version of the Towers of Hanoi MEXPR program shown on page 162. Now let us try something more ambitious:

```
(hanoi '#20)
```

Maybe, this was a bit *too* ambitious, but pressing `Control` and `c` will stop the program:

```
^C
* hanoi.l: 6: append2: interrupted
* Trace: fold2 h h h h h h h h h
```

And once again with a smaller value:

```
(hanoi '#14)
=> ...lots of output...
```

In case you just wanted to know how many moves it takes, there is no need to rerun the program. `Zenlisp` always binds the latest result to the variable `**` after printing it on the REPL, so you can just type:

```
(length **)
=> '#16383
```

And if you want to know why the `hanoi` function is so slow, try this:

```
(gc)
=> ' (#76285 #63103)
```

The first number is the number of unused “nodes” and the second one is the number of used “nodes”. A *node* is an abstract unit which `zenlisp` uses to allocate memory. When the number of used nodes is larger than the number of free nodes (or close to that number), the interpreter will start to spend significant time trying to reclaim unused memory. Running the interpreter with a larger memory pool will give the program a performance boost:

```
(quit)
% zenlisp -b 1024K
zenlisp 2008-02-02 by Nils M Holm
(load hanoi)
(hanoi '#14)
=> ...lots of output...
(gc)
=> ' (#993789 #63098)
```


A.5 zenlisp for the experienced schemer

Zenlisp is very much like a small subset of the *Scheme* programming language as defined in the Revised⁵ Report on the Algorithmic Language Scheme (R5RS), but there are some more or less subtle differences. This is a summary of the most important differences.

The only types are the pair and the atom, atoms are symbols or `()`.

The canonical truth value is `:t` and falsity is `:f`.

Bodies are single expressions, there is no `begin`.

Cond must have a default clause.

`()` does not have to be quoted (but doing so does not hurt).

Predicates do not have a trailing “?”, so you write `(zero x)` instead of `(zero? x)`.

Lists of single-character symbols may be “condensed”: `(x y z) = '#xyz`

Numbers are lists: `(+ '#12 '#34) => '#46`

Special form handlers are first-class objects: `lambda => {internal lambda}`

Apply works fine with special forms: `(apply or '(:f :f :f 'foo)) => 'foo`

Letrec is defined in terms of **let** and **recursive-bind** instead of **let** and `set!`.

Closures have first class environments:

```
(caddr (lambda () x)) => '((x . {void}))
```

All data is immutable, there is no `set!`.

A.6 answers to some questions

Q1, Page 63

Pro: `(headp () x)` should yield `:t` for any x , because all lists have zero leading elements in common.

Contra: `(headp () x)` should yield `:f` for any x , because `(cons () x) ≠ x`.

The confusion arises because the term “head of a list” is used to name two different things: the car part of a pair and the leading elements of a list.

When *headp* was named something like *common-leading-members-p*, things would become clearer. (**Common-leading-members** **() x**) should always yield **:t**.

The bonus question remains: find a better name for *headp* that is shorter than *common-leading-members-p*.

Q2, Page 64

This version of *count* would be confusing, because it would count trailing **()** s of proper lists, so

```
(count '()) => '#1 ; fine
```

but

```
(count '(a b c))    => '#4 ; oops  
(count '(a (b) c)) => '#5 ; oops
```

Q3, Page 66

Yes, *flatten* can be transformed to a function using tail calls exclusively. Any function can be transformed in such a way. In the case of *flatten*, this transformation would not improve efficiency, though, because the function constructs a tree structure and so it has to use some means of structural recursion.

Q4, Page 68

In this particular case, the use of **append** is not critical, because the first argument of **append** does not grow:

```
(trace append) => :t  
(fold-right (lambda (x y z) (list 'op x y z))  
            '0  
            '(a b c)  
            '(d e f))  
+ (append #cf #0)  
+ (append #be ((op c f 0)))  
+ (append #ad ((op b e (op c f 0))))  
=> '(op a d (op b e (op c f 0)))
```

Q5, Page 68

Substitution is not a proper substitute for beta reduction, because it would replace both free and bound variables:

```
(substitute '(list (lambda (x) x)) '((x . #17)))  
=> '(list #17 (lambda (#17) #17))
```

Q6, Page 70

Insertion sort needs about $n^2/2$ steps when sorting an already sorted list of n elements:

```
(load ~nmath)
(load ~isort)
(trace sort)
(isort < '(#1 #2 #3 #4 #5))
+ (sort (#1 #2 #3 #4 #5) ())
+ (sort (#2 #3 #4 #5) (#1)) ; element inserted after 1 step
+ (sort (#3 #4 #5) (#1 #2)) ; element inserted after 2 steps
+ (sort (#4 #5) (#1 #2 #3)) ; element inserted after 3 steps
+ (sort (#5) (#1 #2 #3 #4)) ; element inserted after 4 steps
+ (sort () (#1 #2 #3 #4 #5)) ; element inserted after 5 steps
=> '(#1 #2 #3 #4 #5) ; total = 15 steps
```

It needs n steps to sort a reverse sorted list:

```
(isort < '(#5 #4 #3 #2 #1))
+ (sort (#5 #4 #3 #2 #1) ())
+ (sort (#4 #3 #2 #1) (#5)) ; element inserted after 1 step
+ (sort (#3 #2 #1) (#4 #5)) ; element inserted after 1 step
+ (sort (#2 #1) (#3 #4 #5)) ; element inserted after 1 step
+ (sort (#1) (#2 #3 #4 #5)) ; element inserted after 1 step
+ (sort () (#1 #2 #3 #4 #5)) ; element inserted after 1 step
=> '(#1 #2 #3 #4 #5) ; total = 5 steps
```

The average number of steps required by *isort* is the average of these extreme values.

Because the run time of *isort* is not easily predictable within the range limited by these extremes, it is not practicable as a sorting algorithm.

Q7, Page 73

Strictly speaking, most sorting functions use non-strict predicates to sort sets, even if strict predicates are more popular in the real world. For instance,

```
(S < '(#1 #1)) => bottom
```

should hold for any sorting algorithm S , because no strict order can be imposed on a set containing equal members.

So if you want a mathematically correct notation, non-strict predicates are the way to go. If you want to swim with the stream, strict predicates are what you want.

Q8, Page 73

This version of *insert* is stable when using non-strict predicates:

```
(define (insert p x a)
  (letrec
    ((ins
      (lambda (a r)
        (cond ((null a)
              (reverse (cons x r)))
              ((not (p x (car a)))
               (ins (cdr a) (cons (car a) r)))
              (t (append (reverse (cons x r)) a))))))
    (ins a ())))
```

Any sorting algorithm can be converted from stability under strict to stability under non-strict predicates and vice versa by using the following scheme.

In every sorting algorithm there is a point where elements are compared:

```
(cond ((p x y) sort-them)
      (t      already-sorted))
```

This part of code is modified by negating the predicate and swapping the branches:

```
(cond ((not (p x y)) already-sorted)
      (t      sort-them))
```

Q9, Page 77

Because *unsort* picks members from pretty random positions of the source list, it has the same average complexity as *isort*, which inserts members at pretty random positions.

Q10, Page 80

Omitting the clause would turn *for-all* into a predicate.

Q11, Page 83

The complexity of *combine* depends entirely on its second argument: the size of the source set. Iterating the first argument yields a degenerate curve:

```
(map length
  (map (lambda (x) (combine x '#abcde))
       '(#1 #2 #3 #4 #5 #6 #7 #8 #9 #10)))
=> '(#5 #10 #10 #5 #1 #0 #0 #0 #0 #0)
```

The *tails-of* function is used to iterate over sets of different sizes in order to estimate the complexity of *combine*:

```
(map length
  (map (lambda (x) (combine '#5 x))
       (reverse (tails-of '#0123456789abcdef)))))
=> '(#0 #0 #0 #0 #1 #6 #21 #56 #126 #252 #462 #792 #1287 #2002 #3003)
```

zen style programming

Because the distance between the points keeps increasing on the y-axis of the curve, the complexity is worse than linear. It is probably even worse than $O(n^2)$, because $16^2 = 256$ and the above function yields 3003 for a set size of 16. The complexity is probably better than $O(2^n)$, though, because $3003 < 2^{16}$ and the average increase of the value is less than two times the previous value.

To find out whether *combine* exhibits polynomial behavior with a large exponent or exponential behavior with a small exponent, more precise analysis are required.

Because *combine* uses structural recursion (recursion through **map** is a dead giveaway for structural recursion), exponential complexity seems highly probable, though.

A similar approach can be used to estimate the complexity of *combine**.

Q12, Page 85

Here is the modified *permutations* function with the additional clause rendered in bodface characters:

```
(define (permutations set)
  (cond
    ((null set) ())
    ((null (cdr set)) (list set))
    ((null (cddr set)) (rotations set))
    (t (apply append
      (map (lambda (rotn)
        (map (lambda (x)
          (cons (car rotn) x))
          (permutations (cdr rotn))))
        (rotations set))))))
```

Yes, the modification makes sense, because it reduces the average run time of *permutations* to about 67% of the original version.²⁶

No matter how clever this optimization is, the complexity of *permutations* remains unchanged, because $n!$ permutations still have to be created for a set of n elements:

```
(map length
  (map permutations
    (reverse (tails-of '#abcdefg))))
=> '(#1 #2 #6 #24 #120 #720 #5040)
```

Q13, Page 87

A trivial yet efficient approach to computing $n!$ in zenlisp is:

```
(apply * (iota '#1 n))
```

²⁶ See chapter 2.7 of “Sketchy LISP” (Nils M Holm; Lulu Press, 2008) for a detailed discussion.

It uses the *iota* function from page 86. Can you explain why it is even more efficient than the recursive product method, at least in zenlisp?

Q14, Page 90

Indeed, why not:

```
(filter (lambda (x)
          (or (null (cdr x))
              (apply >= x)))
        (part '#4))
=> '((#1 #1 #1 #1) (#2 #1 #1) (#2 #2) (#3 #1) (#4))
```

Q15, Page 93

$$\begin{aligned}
 3^{(6)} 3 &= 3^{(5)} 3^{(5)} 3 \\
 &= 3^{(5)} 3^{(4)} 3^{(4)} 3 \\
 &= 3^{(5)} 3^{(4)} 3^3 3^3 \\
 &= 3^{(5)} 3^{(4)} 3^6 \\
 &= 3^{(5)} 3^{(4)} 7625597484987
 \end{aligned}$$

The righthand part of the last line in the above equation is once again a power tower of the height 7,625,597,484,987 (as shown on page 92). But this time, that vast number merely describes the number of hyper-5 operations to apply to the factor of 3:

$$\begin{array}{c}
 3^{(5)} \dots\dots\dots (5) 3 \\
 | \qquad \qquad \qquad | \\
 +----- 3^{(4)} 7625597484987 \text{ times } -----+
 \end{array}$$

The number $3^{(6)} 3$ is indeed very hard to describe, even in terms of a power-power tower.

Q16, Page 95

Here is a generator that produces the tails of a list:

```
((generator '#abcdef cdr)) => '(#abcd . {closure ()})
      (next **) => '(#bcd . {closure ()})
      (next **) => '(#cd . {closure ()})
      (next **) => '(#d . {closure ()})
      (next **) => bottom
```

When the end of the list is reached, the generator yields bottom, because **cdr** cannot take the cdr part of (). See the concept of “streams” that is introduced in the following section for a more elegant solution.

zen style programming

Q17, Page 99

Using **fold-r**, the *append-streams** function is easily implemented:

```
(define (append-streams* . a)
  (fold-r append-streams :f a))
```

Applying the function to zero arguments yields the “end of stream” indicator **:f**, just like **(append)** yields the “end of list” indicator **()**.

Q18, Page 99

All non-recursive stream functions (plus *stream* itself) can be applied to infinite streams safely. These functions are

```
stream map-stream filter-stream append-streams
```

Of course appending a finite stream to an infinite stream renders the finite stream inaccessible.

As outlined in the text, *stream->list* is not safe. The *stream-member* function is only safe if a member with the desired property is guaranteed to exist in the stream. The following application of *stream-member* reduces to bottom:

```
(stream-member even
  (stream '#1 id all (lambda (x) (+ '#2 x)) none :f)
  :f)
```

Q19, Page 105

The signature of the signature of a record has only members of the type *symbol*:

```
(record-signature (record '(food apple) '(weight #550) '(vegetarian :t)))
=> '(%record) (food symbol) (weight number) (vegetarian boolean))

(record-signature **)
=> '(%record) (food symbol) (weight symbol) (vegetarian symbol))
```

Because *symbol* is a symbol, this result is a fixed point of the *record-signature* function: passing it to the function will yield the same result over and over again.

Q20, Page 117

The tree structured is formed by the call tree of the program. The inner nodes of the trees are the functions of the parser.

Q21, Page 117

Here is a grammar that gives the unary minus a lower precedence than the power operator, so that

$$-x^2 = -(x^2)$$

Differences to the grammar on page 110 are rendered in boldface characters.

```
<sum> := <term>
      | <term> '+' <sum>
      | <term> '-' <sum>

<term> := <negation>
      | <negation> '*' <term>
      | <negation> <term>
      | <negation> '/' <term>

<negation> := <power>
           | '-' <power>

<power> := <factor>
        | <factor> '^' <power>

<factor> := symbol
        | number
        ; removed the '-' <factor> rule
        | '[' <sum> ']'
```

Q22, Page 121

Here is a version of the *infix* subfunction of *prefix->infix* that places parentheses around all binary operators. The modified *prefix->infix* function does not make use of the *paren* and *add-parens* subfunctions.

Differences to the original version print in boldface characters:

```
(infix
  (lambda (x)
    (cond
      ((numeric-p x)
       (cadr x))
      ((symbol-p x)
       (list x))
      ((and (eq (car x) '-')
            (not (atom (cdr x)))
            (null (cddr x)))
       (append '#- (infix (cadr x))))
      ((and (eq (car x) '[')
            (not (atom (cdr x)))
```


zen style programming

```
(null (cddr x)))
(append '#[ (infix (cadr x)) '#]))
((and (not (atom x))
      (not (atom (cdr x)))
      (not (atom (cddr x)))
      (null (cdddr x))
      (function-p (car x)))
 (append '#[
          (infix (cadr x))
          (list (cdr (assq (car x) ops)))
          (infix (caddr x))
          '#]))
 (t (bottom (list 'syntax 'error: x))))))
```

Adding parentheses to all operations explicitly is useful when translating source code of one language to source code of another language that has different precedence rules.

Q23, Page 121

RPN (reverse polish notation, postfix notation) does not need parentheses, because precedence is expressed by the ordering of operators and operands:

Infix	RPN	zenlisp
((a + b) * c) - d	a b + c * d -	(- (* (+ a b) c) d)
(a + (b * c)) - d	a b c * + d -	(- (+ a (* b c)) d)
a + ((b * c) - d)	a b c * d - +	(+ a (- (* b c) d))
a + (b * (c - d))	a b c d - * +	(+ a (* b (- c d)))
(a + b) * (c - d)	a b + c d - *	(* (+ a b) (- c d))

Note that prefix notation is also unambiguous as long as all operators accept two arguments. Because `zenlisp` functions may be variadic, though, explicit operand grouping is required.

Q24, Page 129

The classes `'#[-x]` and `'#[x-]` match the characters “x” and “-” literally, because neither `x-` nor `-x` is a complete range.

The class `'#[]` matches no characters, so it is a class that never matches. `'#[^]` matches any character except for none, so it is a synonym of `'#_`. However, most real-world regex implementations would probably handle this differently.

Q25, Page 129

Removing one application of **reverse** is all it takes to turn the eager matcher into a lazy matcher:

```
(define (match-star cre s m)
  (letrec
    ((try-choices
      (lambda (c*)
        (cond ((null c*) :f)
              (t (let ((r (match-cre (cdr cre) (caar c*) (cadar c*))))
                   (cond (r (append (reverse m) r))
                         (t (try-choices (cdr c*))))))))))
    (try-choices (make-choices cre s ())))))
```

Make-choices returns the choices in such an order that the shortest match is listed first, so **reverse** is used in the original implementation to give precedence to the longest match. By omitting this **reverse**, the eager matcher becomes a lazy matcher.

Q26, Page 138

Evaluation of special forms cannot be delegated to the outer interpreter, because some special form handlers call **eval** internally. When this happens, the environment of the outer interpreter would be used to look up symbols that are stored in the environment of the inner interpreter.

Q27, Page 138

Because `'%special` is an ordinary symbol, the internal representation of operators can be used in programs:

```
(zeval '(('%special . quote) foo) ()) => 'foo
```

This is only a minor glitch, but you can fix it by using a unique instance [page 55] like `(list '%special)` in the place of the symbol `'%special` to tag special operators, e.g.:

```
(let ((%special (list '%special)))
  (letrec
    ((initial-env
      (list ...
        (cons 'and (cons %special and))
        (cons 'apply (cons %special apply))
        ...)))
    ...))
```

An undesired effect of `'%void` is that it cannot be used as the value of a variable, because doing so would make the variable unbound:

```
(zeval 'x '((x . %void))) => bottom
```

The remedy for this effect is the same as for `'%special`.

A.7 list of figures

Fig. 1 – divide and conquer approach	71
Fig. 2 – run times of sorting algorithms	75
Fig. 3 – complexity of sorting algorithms	76
Fig. 4 – classes of complexity	77
Fig. 5 – syntax tree of $x+y*z$	110
Fig. 6 – right-associative syntax tree of $x-y-z$	112
Fig. 7 – meta-circular interpretation	129
Fig. 8 – the towers of hanoi	162
Fig. 9 – node pool structure	198
Fig. 10 – garbage collection, state 1	208
Fig. 11 – garbage collection, state 2	208
Fig. 12 – garbage collection, state 3	209
Fig. 13 – garbage collection, finished	209
Fig. 14 – symbols, atoms, and atomic nodes	214
Fig. 15 – primitive function structure	216
Fig. 16 – shared list	230
Fig. 17 – numeric tower	275

A.8 list of example programs

combine	81
combine*	81
compose	42
contains	32
count	64
create-conser	37
depth	43
exists	79
ext-sub	60
factorial	86
factors	88
filter	42
flatten	65
fold-left	67
fold-right	67
for-all	80
generator	94

headp	63
hyper	92
insert	69
intersection	24
intersection	24
intersection*	25
intersection-list	24
iota	86
isort	70
lambda-rename	59
lambda-rename	59
let->lambda	57
list->set	78
make-partitions	90
map-car	44
map-car-i	61
mergesort	72
ngcd	48
non-empty-list	26
nth	64
palindrome	12
part	89
partition	66
prefix->infix	117
re-compile	122
re-match	122
remove	43
replace	30
rotate	84
stream	95
subst-vars	61
substitute	68
tailp	63
transpose	93
union	78
unlet	58
unsort	73
zeval	130

A.9 code license

Don't worry, be happy.

Frankly, life's too short to deal with legal stuff, so

- do what ever you want with the code in this book;
- if the code doesn't work, don't blame me.

Disclaimer

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

index

%...|

%d+ function 278
%d- function 278
%d< function 278
%decay function 295
%equalize function 296
%least-terms function 295
%nth-item function 277
%void (zeval) 131, 133
' 9, 134
() 15, 27, 64
* function 47, 285, 292, 300
** 307, 313
diffs-of-digits structure 277
epsilon 52, 300
sums-of-digits structure 277
+ function 47, 285, 292, 300
- function 50, 285, 292, 300
-> operator 13
/ function 51, 300
:= operator (BNF) 107
:f 13
; 19
< function 49, 285, 292, 300
<= function 49, 285, 292, 300
= function 49, 285, 300
== goal (AMK) 166, 188
=> operator 9, 220, 311
> function 49, 285, 292, 300
>= function 49, 285, 292, 300
_, n (AMK) 171, 184
_rdch() 216
{ } 33
| operator (BNF) 107
Q
abs function 292, 300
actual argument 12
actual argument list (MEXPRC) 150
add-parens example (infix) 119
add_primitive() 215
add_special() 215
add_symbol() 215

alist 41, 226
all example (streams) 96
all goal (AMK) 169, 172, 189
alloc() 211
alloc3() 211
Allocations 201
amk 165
and keyword 29, 230, 132, 303, 305
annotation (BNF) 141
anonymous function 33
another micro kanren 165
answer (AMK) 165
any goal (AMK) 168, 172, 175, 188
append function 10, 272, 305
append-stream example (streams) 99
appendo goal 165
appendo goal (AMK) 177
apply function 25, 231, 303, 305
Arg_stack 200
argument 10, 11
argument list 12
arithmetic-iterator function 53, 274
ArrowLISP 276
asave() 212
assert-record-type example 104
assoc function 41, 272, 305
association list 41, 226
associativity 46, 112
assq function 41, 272, 305
asymmetric predicate 69
atom 214
atom function 18, 224, 306
ATOM_FLAG 197, 209, 214
atomic 18
atomic node 214
atomic() 213
aunsave() 212
b
backtracking (AMK) 181
backtracking 127, 163
Backus Normal Form 107
bad_argument_list() 222

zen style programming

Batch 265
batch mode 265
before-p example (regex) 123
beta reduction 37, 61, 135
big-O notation 76
bignum arithmetics 48
`bind_args()` 247
`Bind_stack` 200
binding 10, 247
binding (AMK) 167
binding stack 225
BNF 107, 141
body (**cond**) 13
body (function) 11
body (**let**) 35
bottom 10, 48, 192
bottom function 30, 227, 305
bottom preservation 30
bottom preservation 192
bound variable 34
`Bound_vars` 200, 235
box notation 214
`bsave()` 212
`bunsave()` 212
C
`caar()...caddr()` 204
caar...caddr function 17, 270, 305
call by name 20, 216, 229
call by name (AMK) 172
car function 15, 223, 305
car-of-rest example (parser) 114
caro goal (AMK) 170
case (**cond**) 16
`catch_int()` 267
cdr function 15, 223, 305
cdr-of-rest example (parser) 114
cdro goal (AMK) 170
character class (regex) 122
character set (regex) 123
choice goal (AMK) 190
circular value (AMK) 190
class (regex) 126
clause (**cond**) 13, 132
`clear_stats()` 260
closed world assumption 179
closure 33, 37, 225, 236, 279
closure-form pseudo function 40, 242, 307
`Closure_form` 201, 256
code synthesis 117, 147
`collect_free_vars()` 236
Collections 201
combination 80, 83
combine example 81
*combine** example 81
comment 19
commutativity 121
compiler 139
complexity 75
compose example 42
concatenation operator (BNF) 140
cond keyword (AMK) 172, 175, 181
cond keyword 13, 233, 132, 303, 305
`cond_eval_clause()` 233
`cond_get_pred()` 232
`cond_setup()` 232
condensed list 13, 19
condition 13
conditional expression (MEXPRC) 153
conditional reduction 13
cons 22
cons function 16, 222, 55, 305
conso goal (AMK) 170
constant space 21
contains example 32
continue flag 229
`copy_bindings()` 262
count example 64
`count()` 207
counter 196
counter functions 207
`counter_to_string()` 207
create-conser example 37
`Current_path` 199
cutting (AMK) 180
d
datum 10, 19
decimal full adder 278
`DEFAULT_IMAGE` 196
`DEFAULT_NODES` 195
define keyword 9, 234, 20, 35, 304

`define_function()` 234
defined function 304, 228
delimiter 218
denominator function 51, 294
depth example 43
depth-first traversal 110
Deutsch/Schorr/Waite 209
divide and conquer 70
divide function 48, 285, 292
DOT 197
dotted list 18
dotted pair 17
dump-image pseudo function 307, 243, 310
`dump_image()` 242
dynamic scoping 38, 234
e
eager matching 123
effect 54
eight queens 163
`eliminate_tail_calls()` 249
empty environment 279
empty list 15, 27
`Env_stack` 200
environment 35, 133
environment stack 225
EOT 197
eq function 15, 223, 26, 51, 306
eqo goal (AMK) 172, 192
equal function 28, 30, 272, 305
equal function (records) 102
equality 27
equality (records) 101
`equals()` 220
Error 199
error message 196
error reporting 205
`error()` 205
`Error_context` 196
`Error_flag` 199
eta conversion (AMK) 172
eval function 56, 235, 61, 133, 305
`eval()` 249
`Eval_level` 200
evaluator 197, 247
evaluator state 197, 229

`Evaluator_states` 197
even function 283
existential quantor 80
exists example 79
`expand_path()` 244
`Expanded_path` 199
explicit type checking 104
explode function 59, 224, 306
`explode_string()` 219
expr example (parser) 114
expression 10, 20
expt function 48, 285, 292, 300
ext-sub example 60
f
factor (MEXPRC) 154
factor example (parser) 114
factorial example 86
factors example 88
fail goal (AMK) 165, 169, 186
falsity 13
fat recursion 87
`fatal()` 206
`Fatal_flag` 199
field (records) 99
filter 98
filter example 42
filter-stream example (streams) 98
filtero example (AMK) 178
`find_symbol()` 213
first-match strategy 122
firsto example (AMK) 180
`fix_all_closures()` 227
`fix_cached_closures()` 225
`fix_closures_of()` 226
fixed point 320
`flat_copy()` 231
flatten example 65
fold function 45, 54, 271, 306
fold-left example 67
fold-r function 46, 271, 306
fold-right example 67
folding 45
for-all example 80
formal argument 11
formal language 106

zen style programming

fragment (MEXPRC) 144
Frame 200
free variable 34
Freelist 199
fresh variable (AMK) 168, 187
function 10, 16, 33, 54
function (AMK) 174
function application 231
function application (MEXPRC) 151
function body 11
function call 21
function composition (AMK) 171
function conversion (AMK) 174
function prototype 222
function term 11
Function_name 200
g
garbage collection 197, 208
gc function 307, 228, 313
`gc()` 210
GC_stats 265
gcd 48
gcd function 48, 285, 292
general case 16
generator 94
generator example 94
`get_opt_val()` 265
`get_options()` 266
`get_source_dir()` 243
global definition 20, 35
goal (AMK) 165, 170
grammar (BNF) 109
greatest common divisor 48, 284
h
hack 250
head (list) 16
headp example 63
`help()` 265
higher-order function 42
higher-prec-p example (infix) 118
hyper example 92
hyper operator 91
hyper-exponential complexity 93
i
i* function 290
i+ function 288
i- function 289
i-abs function 288
i-divide function 290
i-expt function 291
i-gcd function 292
i-integer function 287
i-lcm function 292
i-max function 292
i-min function 292
i-natural function 287
i-negate function 288
i-negative function 288
i-normalize function 287
i-one function 288
i-quotient function 291
i-remainder function 291
i-sqrt function 292
i-zero function 288
i< function 289
i<= function 290
i= function 290
i> function 290
i>= function 290
id function 270, 306
identity 26
Image[] 265
Image_vars[] 242
imath package 50, 286
implode function 59, 224, 306
improper list 18
Infile 199
infinite data structure 94
infinite lookahead 159
infix example (infix) 120
`init()` 268
`init1()` 258
`init2()` 259
initial environment (zeval) 130
initialization 258
inner context 36
Input 199
insert example 69
insertion sort 70
integer function 53, 292, 300

330

integer-p function 52, 287
interpretation 129, 195
interrupt 262
intersection example 24, 78
*intersection** example 25
intersection-list example 24
iota example 86
is_alist() 226
is_bound() 235
is_delimiter() 218
is_list_of_symbols() 233
isort example 70
iter package 274
j
juiceo example (AMK) 181
k
keyword 20
l
lambda calculus 38
lambda function (MEXPRC) 152
lambda keyword 33, 237, 134, 303, 304
lambda-rename example 59, 61
language 106
lazy matching 123
lazy structure 94
lcm function 285, 292
least common multiple 284
least terms 295
least type 53
left recursion (parser) 112
left-associativity 46
left-associativity 155
lefto example (AMK) 184
length function 47, 284
let keyword 35, 240, 39, 135, 303, 304
let->lambda example 57
let_bind() 240
let_eval_arg() 239
let_finish() 240
let_next_binding() 238
let_setup() 238
letrec keyword 40, 241, 135, 303, 304
lexeme 106, 144
lexical analysis 144
lexical context 35, 236
lexical scoping 37
Lexical_env 200, 235
limit function 285
Line 199
linear recursion 21
list 10, 13, 17, 19, 63, 69
list function 11, 24, 270, 306
list->record example (records) 100
list->set example 78
list->stream example (streams) 97
listp function 17, 273, 306
literal list (MEXPRC) 149
load pseudo function 307, 245, 309
load() 244
Load_level 200
locality 12, 35
logical AND (AMK) 169
logical equivalence 51
logical falsity 13
logical OR (AMK) 169
logical truth 13
longest-match-first strategy 122
look-ahead token 149
m
m-expression 139
m-expression grammar 140
m-expression grammar 141
make-partitions example 90
make-rational function 294
make_closure() 236
make_lexical_env() 236
map function 43, 58, 273, 306
map-car example 44
map-car-i example 61
map-stream example (streams) 97
mapping 43, 97
mark() 208
MARK_FLAG 197, 208
math functions 274, 308
MATOM 197
matrix transposition 93
max function 285, 292, 300
Max_atoms_used 201
MAX_PATH_LEN 197
Max_trace 201

zen style programming

MBETA 197, 249, 252
MBIND 197
MBINR 197
McCarthy, John 139
MCOND 197
MCONJ 197
MDISJ 197
mem-p function (AMK) 171
member 11
member function 28, 273, 306
memo goal (AMK) 171, 173
memory management 208
memq function 15, 273, 306
memqo example (AMK) 174
mergesort example 72
meta expression 139
meta function 40, 309
meta-circular interpretation 129
min function 285, 292, 300
minimal set of functions 130
MINIMUM_NODES 196
ML programming language 99
MLETR 197
MLIST 197
Mode_stack 200
modulus 291
msave() 212
munsave() 212
mutual recursion 39
n
n queens 163
n* function 281
n+ function 280
n- function 281
n-combination 81
n-divide function 282
n-expt function 283
n-gcd function 284
n-lcm function 284
n-max function 285
n-min function 285
n-natural function 279
n-normalize function 279
n-one function 281
n-quotient function 283
n-remainder function 283
n-sqrt function 284
n-zero function 281
n< function 279
n<= function 279
n= function 280
n> function 279
n>= function 279
N_PRIMITIVES 201
N_SPECIALS 201
natural function 53, 285, 292, 300
natural number 47
natural-p function 52, 278
neg goal (AMK) 179, 190
negate function 292, 300
negation (AMK) 179
negative function 51, 292, 300
neq function 45, 270, 306
next example (streams) 96
nexto example (AMK) 185
ngcd example 48
NIL 197
nil 15
nl() 205
nmath package 43, 47, 274
NO_EXPR 196, 197
node 196, 198, 313
node pool 198, 313
node, atomic 214
Nodes 265
non-determinism 166
non-empty-list example 26
non-terminal symbol (BNF) 107
none example (streams) 96
normal form 9
normalization 52
not function 43, 270, 307
nth example 64
null function 15, 54, 270, 306
nullo goal (AMK) 179
number example (parser) 113
number-p function 52, 285, 292, 300
numerator function 51, 294
numeric predicate 49
O

odd function 283
one function 285, 292, 300
one goal (AMK) 180, 189
operator 155
or keyword 29, 241, 134, 303, 305
order of evaluation 12
outer context 36
Output 199
p
P_ATOM 201
P_BOTTOM 201
P_CAR 201
P_CDR 201
P_CONS 201
P_DEFINED 201
P_EQ 201
P_EXPLODE 201
P_GC 201
P_IMPLODE 201
P_QUIT 201
P_RECURSIVE_BIND 201
P_SYMBOLS 201
P_VERIFY_ARROWS 201
package 62
pair 17
paio goal (AMK) 179
palindrome example 12
parameterized goal (AMK) 170, 179
Paren_level 200
parse tree 109
parser 106, 147
parsing conflict 159
part example 89
partial reduction 13
partition (number theory) 89
partition example 66
pass example (streams) 97
pcbn variable 229
permutation 83
pmode variable 229
pointer reversal 209
pointer reversal algorithm 208
Pool_size 198
power example (parser) 115
power tower 91
power-power 91
pr () 205
precedence 109
predicate 15, 49, 69
predicate (AMK) 171
predicate (**cond**) 13
predicate conversion (AMK) 171
predicate-iterator function 45, 56, 274
prefix notation 110
prefix->infix example 117
prime factorization 88
primitive function 33, 201, 222
primitive operation handler 222
primitive () 229
Primitives [] 201
print () 256
print_call_trace () 205
print_closure () 256
print_condensed_list () 255
print_license () 266
print_primitive () 256
print_quoted_form () 255
print_trace () 249
printer 255
prnum () 205
production 107
program 20
program (formal) 107
program structure (MEXPRC) 147
protected symbol 279
pseudo function 13, 20, 30, 55, 201
q
query (AMK) 165
quicksort 70
quine 57
quit pseudo function 307, 228, 311
quotation 57, 19
quote 9
quote keyword 19, 241, 134, 304
quote () 219
Quotedprint 201
quotient function 48, 285, 292
r
r* function 297
r+ function 297

zen style programming

r- function 297
r-abs function 296
r-expt function 299
r-integer function 296
r-max function 299
r-min function 299
r-natural function 296
r-negate function 299
r-negative function 299
r-normalize function 295
r-number-p function 294
r-one function 295
r-sqrt function 300
r-zero function 295
r/ function 298
r< function 298
r<= function 298
r= function 298
r> function 298
r>= function 298
R_PAREN 197
rational function 53, 295
rational-p function 52, 294
rdch() 216
re-compile example 122
re-compile example (regex) 125
re-match example 122
re-match example (regex) 128
read-eval-print loop 267
read_condensed() 218
read_eval_loop() 264
read_list() 217
read_symbol() 219
reader 217, 216
record 99
record example (records) 100
record->list example (records) 101
record-equal example (records) 101
record-field example (records) 101
record-ref example (records) 101
record-set example (records) 103
record-signature example (records) 102
record-type-matches-p example 104
recursion 16, 31, 39, 87, 108, 225
recursion (AMK) 171, 174, 177
recursive descent parser 113
recursive function 40
recursive product 86
recursive-bind function 41, 227, 304
reduction 9, 13
reduction step 74
Reductions 201
regular expression 121
reification (AMK) 171
reified variable (AMK) 171
Rejected 199, 216
remainder function 48, 285, 292
remove example 43
REPL 267, 311
repl() 267
replace example 30
require function 47, 273, 307, 310
reset_counter() 207
reset_state() 258
rest example (parser) 114
restore_bindings() 263
reverse function 20, 272, 306
reverse polish notation 110, 322
reverse_in_situ() 239
right recursion (parser) 111
right-associativity 156
rmath package 51, 293
rmemqo example (AMK) 176
Root[] 200, 210
rotate example 84
RPN 110, 322
rule (BNF) 108
run* function (AMK) 165, 192
Russel, Steve 139
S
s-expression 139
S_bottom 201
S_closure 201
S_false 201
S_lambda 201
S_last 201
S_primitive 201
S_quote 201
S_special 201
S_special_cbv 201

S_true 201
S_void 201
Safe_symbols 200, 262
save() 212
scanner 147
Scheme 314
semantics, semi-formal (BNF) 141
sentence 106
set 78
setup_and_or() 230
SF_AND 201
SF_APPLY 201
SF_CLOSURE_FORM 201
SF_COND 201
SF_DEFINE 201
SF_DUMP_IMAGE 201
SF_EVAL 201
SF_LAMBDA 201
SF_LET 201
SF_LETREC 201
SF_LOAD 201
SF_OR 201
SF_QUOTE 201
SF_STATS 201
SF_TRACE 201
shallow binding 236
short circuit reduction 29
shortest match first 123
side effect 20, 54, 95
signature (records) 102
sorted list 69
Source_dir 199
special form 201, 229
special form handler 229
special() 247
Specials[] 201
spine (list) 231
sqrt function 51, 285, 292, 300
stable sort 73
Stack 200
Stack_bottom 200
Stat_flag 201
stats pseudo function 74, 246, 307, 310
stopping programs 313
stream 95
stream example 95
stream->list example (streams) 97
stream-member example (streams) 97
string_to_symbol() 214
structural recursion 31, 58, 65
subgoal (AMK) 168
subst-vars example 61
substitute example 68
substitution (AMK) 187
succeed goal (AMK) 165, 186
suffix notation 110
sum example (parser) 116
SWAP_FLAG 197, 208
symbol 9, 214, 59
symbol example (parser) 114
symbol table 200
symbol table 213
symbol-p example (parser) 113
SYMBOL_LEN 197
symbol_to_string() 215
symbolic() 214
Symbols 200
symbols function 307, 228
syntax analysis 106, 147
syntax tree 109
syntax-directed compilation 147
†
tag (records) 99
tail (list) 16
tail call 21
tail position 21
tail recursion 22, 137, 303
tailp example 63
term (function) 11
term (**let**) 35
term example (parser) 115
terminal symbol (BNF) 107
Tmp 199
Tmp2 199
Tmp_car 199
Tmp_cdr 199
token 144
towers of hanoi 162, 313
trace pseudo function 307, 246, 310
Traced_fn 200

zen style programming

transpose example 93
trivial case 16
truth 13
type (records) 99
type checking 27, 53, 99, 104
type-of example (records) 101
U
unbind_args() 237
undefined 10
underspecification (AMK) 184
unification (AMK) 166
union example 78
union of knowledge (AMK) 169
unique instance 55
universal quantor 80
unlet example 58
unreadable form 33
unreadable() 220
unsave() 212
unsort example 73
usage() 265
USE() 202
V
value example (streams) 96
var function (AMK) 166, 187
variable 9, 34
variable (AMK) 166, 168, 171, 176
variable (function) 11
variable (MEXPRC) 151
variadic function 24, 47
Verbose_GC 201
verify() 220
verify-arrows function 307, 229, 311
Verify_arrows 201
W
wrong_arg_count() 222
Z
z_and() 230
z_apply() 231
z_atom() 224
z_bottom() 227
z_car() 223
z_cdr() 223
z_closure_form() 242
z_cond() 233
z_cons() 222
z_define() 234
z_defined() 228
z_dump_image() 243
z_eq() 223
z_eval() 235
z_explode() 224
z_gc() 228
z_implode() 224
z_lambda() 237
z_let() 240
z_letrec() 241
z_load() 245
z_or() 241
z_quit() 228
z_quote() 241
z_recursive_bind() 227
z_stats() 246
z_symbols() 228
z_trace() 246
z_verify_arrows() 229
zebra example (AMK) 185
zebra puzzle 182
zen_eval() 263
zen_fini() 262
zen_init() 262
zen_license() 264
zen_load_image() 260
zen_print() 262
zen_print_error() 206
zen_read() 262
zen_stop() 262
zero function 48, 285, 292, 300
zeval example 130

THE TAO OF ROARK

Variations on a Theme from Ayn Rand

by Peter Saint-Andre

1. The Courage to Face a Lifetime

A young man rode his bicycle down a forgotten trail through the hills of Pennsylvania. The brilliant spring sun warmed him like a conscious caress. The leaves and trees and rocks called to him of the hope and promise of life on this earth. Alone in the wilderness, he felt the fresh wonder of an untouched world, where joy and reason and meaning were not only possible but a simple human birthright.

Some wondrous music of exaltation played in his head, the self-contained joy of endless variations spun out by an inexhaustible imagination. Yet in his life so far he had found precious few words or deeds or thoughts among the acts of men to match the meaning of that music. Not the work of man as a degradation of nature, but as an improvement upon given materials that could fulfill the potential of the earth. Not masters and slaves, but a free and independent life of mutual respect and voluntary interaction, without pain or fear or guilt. Not happiness and achievement served to him by others, but the straightforward sight of joy and reason and meaning made real, which would inspire in him the courage to create his own happiness and achievement.

He could give no name to the thing he sought.

He yearned for an exalted experience of life — but he was told that exaltation is reserved for things not of this earth.

He wanted human activity to be a higher step: something noble that he could respect, even something sacred that he could worship — but he was told that the only nobility and the only proper objects of worship exist above and beyond the sphere of the merely human.

He longed to witness a spark of the divine in his fellow man, and to nurture that spark in himself — but he was told that aspiring to a share in the divine is the height of arrogance.

He hoped to find a way of life animated by a natural reverence for man and this earth — but he was told that the only path to spirituality lies in turning away from this life toward a supposed life after death.

He wished for some sign of what he sought, some guidepost on the road to joy and reason and meaning — but what he sought seemed perpetually just beyond his grasp.

The boy pedalled on through the quiet hills, revelling in the solitude and wondering about his future with the combination of agonized confusion, wistful longing, and passionate expectation that only youth can bring. On the trail ahead he saw a blue hole of open sky where the ridge ended and a valley began. He closed his eyes for a moment, suspending his sense of reality in the strange hope that at the top of the ridge he would find unobstructed sky above and below him.

When he reached the edge he opened his eyes to the most wondrous creation he had ever seen — a valley dotted with small homes that honored the earth and improved upon it by growing organically out of the ground, completing the unplanned beauty of the hills with an even greater beauty of human achievement and fulfillment.

Only after a long while did he notice a man sitting nearby — the man who had made this place real by designing the homes in the valley. Little did the man know that he had given the boy something beyond mere stone and glass: the courage to face a lifetime.

2. The Boy on the Bicycle

I was the boy on the bicycle.

Perhaps you were, too. Perhaps you, too, did something like ride your bicycle down a forgotten trail through the hills of Pennsylvania, wondering if you would find joy and reason and meaning in life. Perhaps you, too, embraced the solitude of your own companionship, treasuring each quiet hour of reflection in a noisy world, loving the very fact of being alone and alive, feeling an unbearable tenderness for the sight of this beautiful earth, breathing in the irrepeatable singularity of your own personhood like great gulps of free fresh air, hungering for all the outstretched possibilities of what you might become — yet daunted by the enormity of the gap between your present and your future, and therefore seeking signposts on the road to the kind of life and character you could in the end look back upon with the pride and honor of a job well done.

Perhaps in your seeking along that lonely path you came upon a novel entitled *The Fountainhead*. For a few days or weeks or months, it changed everything. Then you read the book again, perhaps a few times — challenged in your thinking, stirred in your emotions, deflected onto a new course, imbued with a new fire, transported by a comprehensive vision of life as it might be and ought to be.

Perhaps, after the blinding flash of your first encounter with *The Fountainhead* had mellowed to the warm glow of enhanced awareness, new questions and challenges arose. Is this vision real? Can it be achieved in a world where joy and reason and meaning seem all too rare and elusive? Can I integrate these insights into my own life without submerging my individuality under a flood of ideas and abstractions that, however compelling, were created by someone else? Is Ayn Rand's philosophy of Objectivism not only an intellectual tradition of philosophical analysis and political advocacy, but also a wisdom tradition of spiritual maturity and personal enlightenment?

I, too, have asked these questions and faced these challenges. After more than thirty years of reflecting on *The Fountainhead*, I think that I have gained some

hard-won wisdom regarding the search for joy and reason and meaning in life. I have tried my best to distill and condense that wisdom into this short book.

That I have done so through a set of variations on a theme from *The Fountainhead* no more makes me a spokesman for Ayn Rand than the *Rhapsody on a Theme of Paganini* made Rachmaninoff a spokesman for his Italian predecessor. The theme here is Rand's, but the voice in the variations is my own — a salute to Rand across the chasm of time.

However, the model for this work is not the orchestral extravagance and lush harmonies of Rachmaninoff's *Rhapsody* but the single instrument and contrapuntal austerity of Bach's *Goldberg Variations*. This two-part introduction, a paean to joy and reason and meaning, is the songful aria that sets the tone. In what follows, I explore many variations on that theme; yet, as with Bach, the variations are built not on the melody of the sarabande but on the prosaic, unnoticed, but foundational bass line — with canons and fugues and arabesques sometimes taking the music far from the original notes. After these harmonic excursions, the aria reappears in several restatements of the theme: a hymn to love for existence, the light within, and the meaning of life in what I call the Tao of Roark.

I have worked to maintain a light touch at the keyboard, serious but not preachy, because it is not my place to tell anyone what they "should" or "ought" to think or choose or do or feel. I have presented the Tao of Roark in the first person to make it clear that these are my thoughts, and that I would not dare to replace your own processes of observing, experiencing, thinking, and reflecting. Indeed, at root, I have written this book almost entirely for myself: to determine how I want to live my life, to clarify for myself what I mean by a philosophy for living on earth, to select the values that I deem most important, to enlighten myself about what really matters in life, to inspire myself to reach the highest form of excellence, to create and enjoy something beautiful and uplifting in a world that is too often ugly and small.

Despite the fact that this book is something private and precious and intimately personal, I have chosen to make it public in the hopes that you too will find in these pages some wisdom for yourself.

3. Joy

Howard Roark laughed.

When faced with expulsion from engineering school and the end of his dream of becoming an architect, he didn't whine or complain. He didn't get angry. He didn't blame his misfortune on the government or the schools or the culture at large. He didn't plead with the dean for reinstatement. He didn't worry or fret. He didn't collapse in fear and despair over his career prospects. He didn't even start thinking and planning about what to do next.

No.

Instead, he went for a long walk to his favorite swimming hole, took off his clothes, and dove down into the cool deep waters to enjoy himself and relax.

And, because he wanted to, he laughed.

4. Reason

Working like a convict in the unbearable heat and dust and noise of a granite quarry in midsummer, Howard Roark glanced up to see the incongruous sight of an elegantly dressed woman on the cliff's edge above him. Their eyes met and immediately he knew with intimate, wordless, flagrant understanding that he meant more to her than any man she had ever met, that he caused in her an overwhelming feeling of both shame and pleasure, that she wanted him to take ownership of her in the most masterful, degrading, scornful way possible. From that first glance, they shared a secret, unspoken understanding that she was openly inviting him to rape her.

This may be many things — passion, fire, drama, force, power, will, intuition, insight, projection, mania, lust, intoxication, infatuation, madness — but it is not reason.

Reason is what Roark displayed in his buildings: their logical economy of plan, their organic integration with the site, their crystalline efficiency, their supreme respect for the inhabitants, their comprehensive integrity of design. Unfortunately, it is much more difficult to build those qualities into human relationships and into my own character than into stone and glass.

5. Meaning

Howard Roark and Gail Wynand walked together at the crest of a hill on Wynand's estate in Connecticut. Roark tore a thick branch from one of the trees, grabbed both ends, and bent it slowly into an arc. And he said: "Now I can make what I want of it: a bow, a spear, a cane, a railing. That is the meaning of life."

Wynand, seeing Roark's wrists and knuckles tensed tightly against the resistance of the living green wood, recognized immediately the meaning of his own life. So he asked: "Your strength?"

But strength and power were not the meaning of life for Roark, whose singular focus was to create uniquely original and integrated buildings that would change the shape of things on this earth, for himself and for no one else. So he answered: "Your work."

6. The Soul

In her marriage to Peter Keating, Dominique Francon was perfectly selfless: she displayed no initiative, no desire, no independence, no will, no self, no soul. But what is the self or the soul? It is the thing inside me that thinks, values, makes decisions, and feels.

Roark knew this, too. As he said in his courtroom defense, the functions of the self are to think, to judge, to act, and to feel.

Thought, choice, action, and feeling. In these four human powers, properly understood, can be found a complete philosophy of living.

This insight provides the basis of the Tao of Roark — the foundation for my harmonic explorations — the bass line upon which I build my variations.

7. Thought

The power of thought is my ability to use my mind to understand reality.

Thought involves perception, focus, clarity, objectivity, independence, honesty, integrity, a firm foundation in the facts of reality, a subtle perception of the way things are, the passionate pursuit of passionless truth — yet also empathy, understanding, patience, and the ability to grasp the personal context of those I interact with.

Thought is not only logical, intellectual, or mathematical — it can be musical, literary, visual, spatial, mechanical, organizational, social, interpersonal. Its raw materials are curiosity, imagination, creativity, looking ahead as well as looking back, and focusing clearly on what is present before me. It results not just in knowledge but in wisdom, perspective, insight, and enlightenment.

8. Choice

The power of choice is my ability to direct my energy and attention toward what I find interesting and important in life.

Choice means taking responsibility for my thoughts and actions and feelings, directing the course of my life, having and pursuing my own purposes, relying on my own perception of the truth, bowing to no one's will but mine except through my own free assent, honoring the absolute sovereignty of those around me, and pursuing only voluntary interactions in my life and in society.

Choice implies self-respect in the deepest sense: honoring what I hold to be important, having strength of will and the courage of my convictions, giving my attention to what interests me, devoting my life energy to that which matters most, trusting in my evaluations, spending my precious time on tasks that are consistent with my values, doing what brings me happiness, concentrating on ways to create significant value in the world, focusing on what is under my control and ignoring what is not under my control, seeking to master only myself but not anyone else, and never letting go of my vision of what is possible to me.

9. Action

The power of action is my ability to create value in the world.

The domains in which I can create value and achieve something good are many and diverse: my work, my family, my health, my character, my friendships, my community, and my avocations are primary among them.

Achievement, too, takes many forms: I create value not only if I am a great innovator but also if I incrementally improve an existing technique, if I add to the stock of human knowledge and culture, if I provide a valuable service, if I raise good children, if I strengthen the bonds of mutual respect in my friendships or community or society, even if I only preserve and maintain the good things that were created by those who came before me; indeed, I can gain or keep value in relation to any object, creation, service, process, relationship, art, science, technique, or field of human endeavor.

Further, the good is almost infinitely variable, because any positive value is good: whatever is useful, pleasant, efficient, competent, skilled, masterful, beautiful, elegant, logical, clear, healthy, clean, helpful, humane, wise, loving, kind, courageous, independent, rational, honorable, respectful, dignified, tasteful, joyous, exuberant, passionate, spontaneous, creative, inventive, intelligent, honest, direct, strong, fearless, free, voluntary, serene, innocent, blameless, or integrated is, all other things being equal, good and valuable.

To love the creation of value is to have a boundless energy, a joyous restlessness, a deep intrinsic motivation — it is to achieve an effortless flow of action, a seamless harmony of work and play, a state in which my own person vanishes into the background because I am supremely focused on the doing.

10. Feeling

The power of feeling is my ability to experience the emotional meaning of my thoughts, choices, and actions.

Yes or no, for me or against me, positive or negative, life-enhancing or life-threatening, pleasant or painful, a benefit or a cost, a source of joy or of suffering — at root my capacity to feel is a unique source of feedback about the way I live my life. And, because my life is irreducibly individual, I can find that awareness only through my own emotions. The knowledge I gain comes from how I use my power to think, the direction I take comes from how I use my power to choose, and the value I create comes from how I use my power to act. Those achievements have a direct effect on how well I succeed at the task of human living, which I measure fundamentally by my enjoyment of life. Thus joy is not a mere surface phenomenon, but deep and serious: it is the value that all my efforts go to pay for, the cash value of honoring my true interests in thought and choice and action.

Yet holding joy as an ideal does not mean that I refuse to acknowledge painful facts or experiences. Life can hurt, and the reality of loss is all too often with me. The capacity for joy is but the most positive realization of my capacity for feeling and emotion, and I must nurture that more fundamental capacity if I am to find the greatest joy in life. The actions and creations that I value most exhibit an openness to emotional experience. At its best, that experience is positive; but being open to experience means not shrinking from the negative, either.

Further, my emotions are not only positive or negative, on or off, white or black; they can be tremendously subtle. Consider the differences of intensity, depth, and energy between being calm or serene, cheerful or exuberant, satisfied or fulfilled, involved or engaged, interested or passionate, happy or ecstatic. Because emotions are a form of awareness, attending to these subtle differences can create a profusion of color in my life.

Finally, not all of those colors need to be fiery and intense. Although Rand's novels celebrate the highest passions, they also underline the importance of less

ardent emotions: the sense of family that Roark and Henry Cameron feel in performing a daily routine together; the tenderness of Roark silently placing his hand on the shoulder of the night watchman at Cortlandt Homes; the firm sympathy and complete understanding that Roark extends to Steven Mallory when he needs it most; the bonds of trust, good will, and brotherhood that Roark and Mike Donnigan feel for each other; the fact that the employees in Roark's architecture office experience him as warm, approachable, and deeply human; the young, kind, friendly laughter that comes from Roark when he is talking with Peter Keating about their chosen profession; the natural joy and mutual confidence that Mallory, Donnigan, and Roark experience when they are together; the quiet satisfaction that Roark feels about having designed a building (even if, like his Stoddard Temple, it is disfigured beyond recognition).

11. Harmony

Roark's rule of building is this: "Nothing can be reasonable or beautiful unless it's made by one central idea, and that idea sets every detail. A building is alive, like a man. Its integrity is to follow its own truth, its one single theme, and to serve its own single purpose."

Yet the integration Roark describes is more literary and aesthetic — making my life a work of art — than human and practical — being successful at the great task of living.

What kind of integration is possible to me as an individual?

At root, it is the harmony of thought, choice, action, and feeling.

Integrating thought into choice, action, and feeling means that my knowledge is not an idle curiosity or an end in itself. Instead, I use what I learn about the world and myself as a firm basis for the choices I make, the direction I take, the objects and people I attend to, the activities on which I spend my time and energy, the areas in which I focus, the things I judge to be within my span of control or influence, even the feelings and emotions I consider to be healthy and justified.

Integrating choice into thought, action, and feeling means concentrating my efforts at learning in areas that are congruent with my nature and my interests, taking seriously the responsibility to use my mind and weigh the evidence of my senses and draw my own conclusions, focusing my time and energy where I can have a significant impact on the world around me, taking a systematic approach to realizing my values and decisions in action, evaluating what I know and do and feel in the light of what is important to me — and adjusting my direction in life accordingly.

Integrating action into thought, choice, and feeling means connecting what I learn and know back to the practical concerns of living, always preparing myself physically and emotionally for the realization of my ideas and choices in action, studying methods for becoming more productive and creative, increasing my

competence and mastery in my chosen profession and the other pursuits that matter to me, actively investing in friendships and relationships that might bring me joy, choosing values that I can realistically achieve, cultivating thoughts and feelings that lead to successful action and pruning those that don't, focusing my energy and attention on the value that I want to create in life.

Integrating feeling into thought, choice, and action means attending carefully to my emotional reactions, honoring my emotions as a form of awareness that yields evidence about myself and my values, using the possibility of joy as a great motivator for action, valuing the honesty of my feelings as a true indicator of how successful I have been in my thoughts and choices and activities, learning to enjoy that which is good and valuable so that my ideas and values and feelings are a seamless whole.

This depth of inner harmony is hard to achieve. I do not need to also introduce the notion of integrity as a single theme or purpose that sets every detail of my life. As a result, I pursue that which is humanly achievable, not that which is impossible to me.

12. Objectivity

Objectivity is hard.

To be objective is to focus on the way things are, not the way I wish things would be. It is to recognize the facts of reality, no matter how difficult and unpleasant they are. It is, as Thoreau said, to work my way through the mud and slush of opinion, prejudice, tradition, delusion, and appearance, until I come to the hard bottom of rocks in place, which is reality. It is to admit as true only that which corresponds to the facts as I have worked to perceive them, clearly and without illusion. I call this clear perception "the track of truth".

To be objective is to be aware of the many ways I can stray from the track of truth: that I am drawn to evidence confirming what I already believe, that I seek out those who agree with me, that events can prime me to accept ideas that might be in error, that I am overconfident about the extent of my knowledge, that I presume to know when I don't, that I jump to conclusions, that I succumb to the power of symbols, that I am tempted to hew to party lines and cave in to peer pressure, that I follow fads and fashions all too easily, that I overvalue the insights of those within my group, that I want to believe things that are beautiful or exciting or consistent with the rest of my beliefs, that the seductions of ideology can blind me to the facts, that I desire knowledge without process and insight without effort, that few things are more difficult than honoring the considered judgment of my own mind.

To be objective is to know that these snares and traps and idols apply to everyone, but that I especially am not immune from them, so that I must expend great energy to resist them. It is to have the childlike simplicity of accepting events as if they cannot be changed, to recognize what is within my span of control and what is outside of it, to know that I can control and possess only myself, not anyone or anything else. It is to immerse myself in the facts: in science, in history, in statistics and numbers, in the evidence of my senses, in art as a created object. It is to pay close attention to these things, to really see and hear and know them, to think clearly about them without preconceptions.

When I am objective, I stay on the track of truth: I recognize its faintest signs in the undergrowth of physical reality and human culture, I quietly attune myself to its voices and musics and rhythms through all the noise and chatter of society, I can feel its finest textures in my fingertips, I can even sense when the lack of it smells wrong or leaves a bad taste in my mouth. Many are the manifestations of truth, and by turns I must be subtle, direct, serene, and bold to grasp them.

13. Honesty

Honesty is the essence of objectivity in a social context.

Honesty is my recognition that you too are a thinking being, that you too have the same cognitive relationship to the world that I do, that you too have the same desire to know that I do, that both of us are focused on the same reality and thus will likely come to similar conclusions if you and I both use our powers of thought.

My honesty enables you to be more objective. Your honesty enables me to be more objective. Indeed, my honesty also enables me to be more objective, because it instills in me a habit of truthfulness. By being honest with each other, you and I build a shared commitment to recognizing reality and sustain a shared culture of achieving objectivity.

14. Self-Knowledge

Objectivity about myself is hardest of all. Its many meanings are captured in the phrase "Know Yourself", inscribed at the ancient temple of Delphi and part of the core wisdom of classical civilization.

To know myself means to know my measure, my limits, my powers, my abilities, my special talents; to know my strengths and weaknesses; to know my place, my role, my context, my calling; to know what I can and cannot do; to know what I can and cannot be; to know the limits of my knowledge and wisdom, what I know and do not know; to know what I truly want in life; to know the name of my soul, my real identity, my true self; to know how easy it is to sell my soul and how hard it is to keep it; to know human nature; to know divinity.

And to know myself means to know, finally, that it is hard to know all this because self-deception is the easiest thing in the world.

How to attain self-knowledge? There are many paths: knowledge of history, anthropology, sociology, psychology, biology, and evolution; experience of novels, drama, music, poetry, and each of the other arts; exploration of philosophies, religions, and spirituality; observation, experimentation, action, cooperation, and the exercise of all my faculties; mentorship and teaching; love and friendship; meditation, reflection, and solitude.

I walk as many of these paths as I can. They are tools that can help me gain objectivity about myself, but even with these tools in hand I know that it is harder to be honest about myself than about anything else in the world.

15. Responsibility

Responsibility is hard.

To be responsible is to be held answerable and accountable for my actions, because a full, objective account of what has happened needs to include what I did or didn't do.

To be responsible is to understand and accept myself as a cause of what happens in my life. Not necessarily as the only cause, but certainly as the primary cause. It can be difficult to remain objective about the degree to which I am a cause for any given event or its consequences, for I tend to attribute my successes to myself, and my failures to others and to external circumstances.

To be responsible is to govern my thoughts, my choices, my actions, and my feelings. Thus responsibility is a form of self-governance, and a precondition of freedom; for great freedom imposes great responsibility.

Responsibility looks back at what I have done, but it also looks ahead — for if I am responsible then I shall choose and act with the expectation that I might need to answer for what I do. Thus a sense of personal responsibility induces deliberation, caution, prudence, consideration, careful planning, even good manners.

When I am responsible, I walk a consistent path and I have a consistent aim in life. I have integrity, constancy, solidity, coherence, harmony, wholeness. By being true to myself and my principles, I become someone who is worthy of honor, trust, and respect.

There is much work involved in taking full responsibility for my life: I must learn and apply how to succeed in my chosen profession, how to maintain my health, how to save and invest for the future, how to defend myself and my family, how to be a good friend to those I care about, how to build a strong relationship with my partner in life, how to think clearly, how to make good decisions, how to be productive and creative, how to communicate effectively, how to cultivate my inner life, how to exercise self-control and achieve self-mastery, how to

continually improve myself as a human being. Although it is tempting to think that I can depend on someone else to do these things for me — my family, my friends, a service I hire, a company I work for, a government agency, a charity that will help me in my time of need — the harsh truth is these tasks are my responsibility and mine alone.

16. Respect

Respect is the essence of responsibility in a social context.

Respect is my recognition that you too are a choosing being, that you too must make your own decisions and select your own values and achieve your own happiness, that you too have the same desire that I do to succeed in living a fully human life.

If I am a good and worthy person, the prospect of earning my respect challenges you to become more responsible and to live up to your highest standards. If you are a good and worthy person, you inspire greater responsibility and moral ambition in me. Earning respect is not living second-hand, as long as I choose to interact with those I can honestly respect and admire. Calling each other to responsibility is one of the great values of friendship; yet here too my actions can also influence the broader culture of interactions I have with all people, not only those who are dear to me.

There is a beautiful Sanskrit word that captures the core of such respect: *namaste*. It means: "the divine in me honors the divine in you".

17. Self-Trust

Just as I am the easiest person to deceive, so I am the hardest person to trust.

To be trustworthy inside and out, I must have great command of myself, great mastery of my emotions, great loyalty to my principles, great constancy of purpose, great internal discipline. I must keep straight, guide myself, monitor myself, point myself in a consistent direction, set my own path in life, live up to my ideals, and aspire to the highest excellence I can achieve.

To be trustworthy inside and out, I must have a strong moral compass within me, and not merely respond to pressures and sanctions that come from outside of me. I must choose my own principles and command my own laws, for myself alone and for no one else. I must be sovereign, autonomous, and a more strict governor of my own actions than any external force could ever be.

To be trustworthy inside and out, I must do what I want. As Peter Keating observed, this is not the easiest thing in the world but the hardest: to know what I truly want, what is best for me, what is consistent with my highest potential — and then to have the courage of my convictions by working hard to achieve that in my life.

To mine own self be true — this requires deep self-knowledge and great self-discipline. Self-respect, self-esteem, self-consideration, self-love — these are secondary effects, of which a hard-won self-worth is the cause. I must not confuse the cause and the effect.

18. Productivity

Productivity is hard.

To be productive is to create or preserve what is valuable and important, to achieve something significant in my life.

If I am to be productive, I must focus on my priorities, concentrate on high-value activities, channel my values and choices into achievable tasks and concrete actions, and be disciplined about my time and my life — for there is no self-direction without self-discipline.

When I create something of value, I close the gap between wish and fact, between the ideal and the real. By no means does this happen all at once; depending on what I want to achieve, it can take months or years to realize my goals, to make my values real in the world — with many setbacks and obstacles and challenges along the way. Patience and persistence are essential to my success.

Productivity is not a duty, but a desire for something higher and greater in my life: a matter of aspiration, constructive passion, and positive energy applied to the task of making my values real on this earth. At root, productivity is an expression of love, for to be the kind of person who gets things done, I must above all love the doing.

19. Collaboration

Collaboration is the essence of productivity in a social context.

Collaboration is my recognition that you too are an active being, that often you and I can achieve more together than apart, that often the creation of value is not a solitary pursuit but a matter of mutual achievement, focus, discipline, energy, and aspiration — a matter of being drawn together toward a shared goal in a shared pursuit based on shared values.

There are many challenges here. If there is no self-direction without self-discipline, is there also no shared direction without shared discipline? How can I respect your individuality within a group effort? It helps that you have your own domains of expertise and I have mine, that you and I can divide up the work so that you have your tasks and I have mine — but you and I always face the dangers of groupthink, of hierarchy, of power, of holding the organization above the individual. I must continually be mindful of these dangers if you and I are to work together with mutual respect and complete voluntarism.

20. Self-Improvement

My most challenging project is myself.

No area of possible achievement involves greater obstacles, difficulties, and resistances than my own soul. Yet no area offers greater potential rewards.

Self-improvement is soul-improvement: being productive of my character. I can improve my mind, my body, my decisions, my emotions; I can pursue excellence that is intellectual, physical, financial, ethical, professional, cultural, spiritual; I can be morally ambitious, I can define and refine my moral compass; when I make mistakes I can realize them, admit them, and correct them; I can seek discipline and control and mastery over myself, I can focus my self-knowledge and personal responsibility into internal action through superior habits of thought and choice and feeling.

Self-improvement is a matter of improving, not remaking. If I seek to improve, I accept myself as the foundation; I see myself as material for action; I build upon what I already am. I do not raze the site upon which I shall build my life; instead I integrate my building with the site, just as Roark did with the homes he designed.

The primary result of self-improvement is not self-esteem, because a mere feeling about myself is never primary. No, the primary result is something much harder to achieve: self-value and self-worth.

21. Passion

Passion is hard.

To be passionate is to love this earth and everything in it, to love my life, to be devoted to my values and ideals, to be fully engaged with the world, to take my life and my soul seriously, to never give in to despair, to always hold on to a great sense of hope and if necessary to make my own hope, to provide the motive power in my life, to be thankful for all that I experience, to care deeply about making the world a better place through my thoughts and choices and actions.

If I am passionate about my life, I don't float through it — I live with attention, engagement, awareness, curiosity, interest, devotion, energy, enthusiasm, motivation, commitment, depth of feeling, moral ambition, love for my values, love for existence, and reverence for life in all its forms.

22. Compassion

Compassion is the essence of passion in a social context.

Compassion is my recognition that you too are a feeling being, that you too experience the emotional meaning of life in irreducibly individual ways, that you too are capable of pleasure and pain, joy and suffering, triumph and tragedy — that you too have opened yourself to great feeling and to the vulnerability such openness brings.

Can I give you empathy without pity, understanding without condescension, attentiveness without influence, commitment without exclusion? Can I help to bring out the best in you without seeking to direct your life? Can I see what is best for your life without seeking to impose it upon you? Can I treat all people with humanity, some with fellowship, fewer with friendship, fewer still with great love — without falling prey to the traps of in-groups and out-groups, judging without individual understanding, and the false alternative of deification versus demonization? Can I use my broad understanding of life not as a means to ignore your context but as the basis for treating you with presence and perspective? Can I see my relationships as a source of great value without basing my self-worth on the approval of others? Can I give my love precisely because those I love are not my chief reason for living?

These are the challenges of compassion.

23. The Inner Life

The hardest passion to accept is my passion for myself; the most difficult person to love is myself; the attention I most resist is directed within.

To experience life inside myself, I must accept my feelings as signs and signals of what I truly value, I must take clear perception as a precondition of strong feeling, I must listen to myself and train myself to hear, I must attend to the fine gradations of my emotional experience.

The inner life involves true joy in the senses, for seeing and hearing and touching and tasting are deeply human ways to know and love what is.

When I cultivate my inner life, I attend to myself; I am not afraid to be alone, indeed I revel in solitude and I enjoy my own companionship.

Can I be a friend to myself? Can I honor myself? Can I be self-contained yet still reach out to others and to the world at large? Can I grant myself compassion and empathy and understanding while never giving up my drive for self-improvement? Can I love myself in the toughest and tenderest ways possible, and strive above all to be worthy of such reverence? Can I hold onto my dreams, hopes, ideals, aspirations, deepest interests, and inner passions? Can I refuse to float on the surface of life but instead dive deep within myself in a search for individual freedom, personal dignity, spiritual depth, and moral beauty? Can I do all this without pretension, with simplicity and seriousness and humility and a realization of how far I have yet to go in my search for joy and reason and meaning?

24. Inner and Outer

The first integration I can achieve is an inner harmony of thought, choice, action, and feeling. The second integration I can achieve is a harmony between my inner life and my outer life.

My inner life and my outer life are two aspects of the same achievement. To consistently track the truth implies that I am equally honest with myself and with you, that I equally seek knowledge of reality, of other people, and of myself. To consistently honor self-direction means that I choose my own direction in life and also that I respect the direction you have chosen for yourself. To consistently create value implies that I create value in the world through my work, that I create value in my relationships with other people, and that I create value within myself by improving my habits and my character. To consistently experience meaning implies that I am passionate about my own life and compassionate about the lives of others.

If I am to recognize and respect the powers of thought, choice, action, and feeling in myself, I must recognize and respect them in you. Those I interact with, those I work with, those I befriend, those I love, all are thinking, choosing, acting, feeling beings. I must respect their intelligence, autonomy, experiences, activities, emotions, perceptions, insights, choices, and freedom. And further: I have a great opportunity to learn from the people in my life, through mutual work and trade and friendship and love.

The same principles apply whether I am facing inward or facing outward. To live with integrity is to be of one piece, to be faithful in all I do to my best self and to the irreplaceable style of my soul.

This great integration is the harmony of the inner and the outer.

25. Self and Other

Understanding others is knowledge, but understanding myself is enlightenment; mastering others is power, but mastering myself is strength. Therefore in knowledge there is power, but in enlightenment there is strength.

Being selective about others is preference, but being selective about myself is simplicity; experiencing others is pleasure, but experiencing myself is depth. Therefore in preference there is pleasure, but in simplicity there is depth.

Knowledge, power, preference, and pleasure are signs of desire; enlightenment, strength, simplicity, and depth are signs of purpose.

When I focus primarily on controlling others or seeking their approval, I live second-hand and thus I am on the path to desire, dissolution, and dependence. When I focus primarily on improving myself, I live first-hand and thus I am on the path to purpose, integrity, and autonomy.

26. To Think and Not to Think

Honoring the power of thought gives me great enlightenment, but sometimes greater enlightenment comes from not thinking.

Sometimes I have thought everything I can think for now, as when Howard Roark went for a swim at the quarry instead of planning the next phase of his life.

It is pointless to think and plan far beyond the horizon of my lifespan. History provides valuable perspective, but I cannot change the past. Envisioning the future helps me navigate my direction in life, but the future might change so radically that I am better off learning to be flexible than becoming attached to the way I think things will be.

It is more productive to think about what is within my control than to worry and fret about things that are outside of my control. Worrying is not a form of thinking.

Much of what happens is not worthy of my attention. Fads, fashions, celebrities, news, propaganda, advertising, politicians, and the like are mostly meaningless ephemera. If I know how many things are unimportant in the world, I can focus on what is truly important in my life.

27. To Choose and Not to Choose

Honoring the power of choice gives me great simplicity, but sometimes greater simplicity comes from not choosing.

Some choices are already made for me. I am what I am, and the challenge is to become what I can be, not to change myself fundamentally and completely.

Roark is described as a force of nature. Do I try to change the earth or the ocean or the sky? No, but I can harness them to achieve my values. Just so I can harness and direct and master myself. I am a unique combination of talents, interests, sensitivities, capacities, and potentials. Why attempt to overcome these things, when instead I can use them as the strong foundation for building my life?

Yet the things that I choose are the essence of my life. When I choose something, I accept it, affirm it, admire it, let it into my life, give it my living energy, say yes to it in a total, undivided way. It becomes part of me and I become as faithful to it as I am to myself.

28. To Act and Not to Act

Honoring the power of action gives me great mastery, but sometimes greater mastery comes from not taking action.

After Roark received his first commissions, his stream of clients dried up. Strangely, he did nothing. He went to his office every day and sat in silence and inactivity. Did he know that he had to wait patiently for the time when he could succeed?

There is a time for action and a time for inaction. Many things cannot be achieved directly. Can I directly achieve happiness, enlightenment, dignity, beauty? No. These are things that must be built slowly, over time. I can approach them only from the side, not head on. In the *Tao Te Ching*, this is called action through non-action.

29. To Feel and Not to Feel

Honoring the power of feeling gives me great depth, but sometimes greater depth comes from not feeling.

Just as worrying is not a form of thinking, so it is best to cast off many of the negative emotions of life. Yes, I can feel righteous in my anger, realistic in my pessimism, justified in my anxiety, alive in my misery or grief. Yet the essence of life is joy, serenity, optimism, creation, energy, action.

Does this path lead to repression? I once thought so, but now I see how unproductive it is to wallow in negative emotions. Notice how, even with those who wanted to destroy him, Roark did not hate others or fear them or get angry with them or let them cause him suffering, because that which is negative goes down only so deep and does not touch the essence of life. I can experience and accept such things without giving up the fundamental assertiveness of the life force within me, without giving up my positive right to happiness and beauty and fulfillment.

Knowing when not to feel gives me a peculiar sense of freedom, a sense of being light, clean, unpressured, unburdened, self-contained, without ties to all that is ugly and small.

30. Yin and Yang

Howard Roark was not an Objectivist. To live successfully, he found it necessary to balance thinking and not thinking, choosing and not choosing, acting and not acting, feeling and not feeling — to balance what Chinese philosophers call the yin and the yang.

The yang is that which is more rational, Apollonian, objective, public, open, well-known, bright, scientific, logical, explicit, lucid, clear, hard, dry, airy, powerful, active — that which is related to the sky gods, to high mountains, to Olympus.

The yin is that which is more emotional, Dionysian, subjective, private, personal, unknown, shadowy, humanistic, perceptual, implicit, tacit, opaque, soft, damp, watery, solid, yielding, passive — that which is related to the gods of land and water, to things that are earthy and oceanic.

To think clearly and rationally, to choose my values and focus on what I find important, to create great value and reshape the earth in the image of my values, even to be passionate about life — these, for Rand, are facets of the yang. Yet the yang is not everything. There are aspects of life that are irreducibly yin — the kinds of things that are hard to put into words but that instead must simply be experienced: music, painting, sculpture, dance, gardening, nature, manual labor, athletics, exercise, physicality, sensuality, breath, yoga, meditation, introspection, reflection, contemplation, reverence, awareness, perception, the senses, beauty, adornment, pleasure, relaxation, spontaneity, friendship, love.

These phenomena, these manifestations of yin, have their philosophies, too: Taoism, Buddhism, Epicureanism, aestheticism, gnosticism, organicism, naturalism, yogism, spiritualism, and more. These philosophies and practices can help me gain important insights into the meaning of life.

The great challenge is to find unity in diversity, to achieve a harmony of opposites within myself, to attain a balance among the forces and qualities represented by the yin and the yang. This is not easy; indeed it is one of the supreme challenges of living. Yet I cannot climb that great mountain of wisdom if I am the hedgehog who knows only one big thing; instead, I must be the fox who

knows many things and who has many ways of knowing. I must be open to experiencing life, to recognizing what I truly want even if it appears to be at odds with the yang-like philosophy of Ayn Rand.

31. The Architecture of Happiness

The newspaper caption beneath a picture of Howard Roark standing before the Enright House reads: "Are You Happy, Mr. Superman?" The irony is that Mr. Superman is indeed happy, because making his values real by bringing beautiful buildings into the world is, for Roark, a source of the most exalted enjoyment one could imagine.

This level of joy is not mere fun or pleasure, but a deep alignment between my values and their realization in the world — a kind of metaphysical joy or love for existence. Yet is such joy found only in the highest creations of the human spirit? Does joy require in all instances a feeling of man-worship or a heroic sense of life? No. For me, joy is the word that best captures a deeply positive, constructive, humanistic approach to life, work, art, love, family, friendship, and the pursuit of wisdom. This approach to life is built on the assumption that man is born to glory and that happiness is my sacred birthright.

Does honoring the power of thought require me to live up to an explicitly rational view of man and the universe in everything I think and do, or even to forsake passion? No. I live a life of reason when my actions are clear, intelligible, integrated, open to the fundamental human power of understanding myself and the world. But the power of understanding includes perception, imagination, and introspection as well as explicitly conceptual thought. As Jacob Bronowski wrote in his poem *The Abacus and the Rose*, I must "reject the feud of eye and intellect"; reason's hand, far from being cold and clammy, provides the touch that enables both light and heat, both thought and passion, both deep understanding and deep emotion. Joy and reason go hand in hand.

Finally, can meaning be realized only in the loftiest abstractions or most cosmic goals? Does the search for meaning mandate that I must take an explicitly philosophical approach to every aspect of my life? No. Meaning emerges through an interaction between my choices and my actions, in the self-directed achievement of what I have chosen as good or important. But the good and the important are not mere abstractions: they can be as particular as the smile of a friend, the scent of a flower, the sense of a phrase. Individuality extends that far;

and meaning is found not merely in the cosmic and the universal, but also most directly in the concrete, the particular, and the deeply personal: in the activities of my work, in the loving kindness of my family, in the support of my chosen friends, in the fellowship of the communities in which I live and act, in the irreplaceable health of my body, in the character that I build up within me, in my creative pursuits, in the ornaments of life that I enjoy in nature and human culture.

32. The Sovereign Individual

Roark said that thought, choice, action, and feeling are the functions of the self. In order to live a successful human life, I must be independent in my thinking, my choices, my actions, and my feelings. It is this independence that Roark possesses but that Keating lacks.

This independence gives me ultimate power over my own life. Not power over others, but power over myself: power to understand reality, to direct my energy and attention, to create value, to experience meaning.

To be independent in this way is to be a sovereign individual, to be a law-maker for myself, to be a self-governor.

This supreme independence makes me free.

33. A Higher Step

Supreme independence makes me free. But free for what? Is it enough merely to be free, to be without ties to the world, to govern myself in solitude and inactivity?

Roark's way of life says no.

Independence is but a precondition, which frees me to create great value, to make something that is an improvement upon nature instead of a degradation, to produce a higher step that would be impossible without human action in the world. It is this fundamental creativity that Roark possesses but that Toohey lacks.

A higher step is respectful of nature, just as Roark's structures respect the sites upon which they are built. It knows that nature has its own beauty, and it strives to add further beauty that even nature could not provide.

The principle of the higher step is a difficult taskmaster. It is much easier to blast away the granite of a mountain than to work with that granite to build a Heller House or a Monadnock Valley. It is also much easier to blast away the foundations of my personality and remake it in the image of Rand's philosophy than to engage in the more delicate task of self-improvement.

Is my work a higher step above what I inherited from nature and tradition? Are my relationships higher steps above what my family and earlier generations bequeathed to me? Is my soul a higher step above what nature and nurture provided to me?

My independence frees me to create great value. It is up to me whether I make that potential real.

34. The Great Task

Why create value? Does the world deserve that effort from me? Doesn't creating great value place me at the mercy of the world? Wouldn't it be easier to seek power over the world so that it cannot harm me?

Here again, Roark's way of life says no.

In his job interview with Henry Cameron, Roark says he doesn't like the shape of things and that he wants to change that shape through his own efforts, through the application of his own creative power — but not through power over others. This work, expressed in architecture, is the great task of his life. It is just such a great task of value-creation that Roark possesses but that Wynand lacks.

Is it realistic for me to have the goal of reshaping the world in the image of my values? Not directly. But then all Roark did was design some buildings — it's not as if he reshaped the entire world. Instead, he made his values real in the world through the limited yet still significant scope of what was possible to him.

The ethical issue here is not the relative extent of what I can achieve in life compared to famous inventors, scientists, or artists, but the absolute extent of what I can achieve based on my interests, my talents, and the energy I can realistically expend on various projects and relationships during the brief span of my life on this earth. At that level, I too am capable of great things.

35. The Noble Soul

I am capable of great things. But it takes enormous discipline to truly understand the world and myself, to focus my energy and attention on what I find interesting and important, to create value in myself, my relationships, and my projects. It would be much easier to float through life and to depend on the achievements of others.

Yet to do so would be unworthy of a truly human being. If I abdicate my responsibilities and do not live up to my potential, then I forsake the birthright of a glorious, successful human life. Deeply positive thought, choice, action, and feeling are right and proper to a noble soul — and if I do not strive for nobility, why am I here?

36. The Spirit of Youth

Just as the boy on the bicycle was captivated by Monadnock Valley, so *The Fountainhead* appeals deeply to young people who seek joy and reason and meaning in life. Rand's novel is a confirmation of the spirit of youth, capturing the almost-painful sense of expectation with which a young person can enter into the greater world. Think of those in the courtroom as Roark takes the stand at his trial, who for a moment see him as he really is and who feel the same potential in themselves: independent, strong, capable, courageous, benevolent, clean, innocent, fearless, free.

Is that radiant picture an illusion? Do the curiosity and idealism of youth need to give way to a passionless wisdom — energy and enthusiasm to a settled maturity — openness and flexibility to a cautious security — courage and daring to a conservative practicality?

Perhaps not.

Perhaps, instead, true security comes from self-reliance, from the strength of my skills, from my health, from self-control and self-mastery, from limiting my needs and desires to what is natural and becoming of a liberated individual, from a sense of fellowship with chosen friends who honor the same values I do.

Perhaps true practicality comes from cultivating the deepest sense of who I am, from immersion in life, from an unwavering focus on what matters, from knowing the name of my soul.

Perhaps true maturity comes from holding onto the right ideals, from self-respect and respect for others, from a strong sense of personal responsibility, from continually improving myself, from becoming who I am and what I can be.

Perhaps true wisdom comes from the passionate search for passionless truth, from endlessly seeking new experiences, from always seeing the world with fresh eyes, from never succumbing to conventional categories and party lines, from being true and direct in my dealings with self and others.

37. Freedom

The Fountainhead is a novel of freedom. This freedom is not political but personal: the freedom of self-reliance, of skill, of being capable of surviving and thriving in the world, of standing on my own two feet, of moving through life with strength and independence and competence.

This freedom is the positive liberty to do what matters — to create, to produce, to think, to choose, to act, to feel, to live.

This freedom is the liberation of holding nothing back from my life, of being fundamentally open to experience, enlightenment, dignity, depth, and beauty.

This freedom is the true wealth of creativity, of friendship, of love, of knowing what I truly want and doing what I truly love.

This freedom is the result of self-governance, self-mastery, and self-trust — for when I trust myself completely, I do not need to depend on some authority outside of myself and my relationship to reality.

This freedom brings the ultimate security and leads to the ultimate serenity.

Yet this freedom can be approached only from the side. I cannot grasp it directly. It must grow within me and around me, through the decisions and actions I take every day, through the responsibilities I shoulder, through the powers I exercise. The feelings of liberation that I experience are signs of success, and result from the hard work of personal responsibility.

38. Dignity

In *Les Misérables*, Victor Hugo says that work makes one free and thought makes one worthy of freedom. And to be worthy of freedom is to have a fundamental dignity of soul.

Dignity means idealism without partisanship, self-possession without self-importance, purpose without anxiety, perspective without detachment. It means being thoughtful but not argumentative, steadfast but not stolid, serious but joyful, patient but not complacent, respectful but approachable, quiet but not numb, active but not frenzied. It means strength, poise, style, and grace.

There is no substitute for personal dignity, and no standard of dignity except independence. Not the surface independence of fads and fashions, but independence where it matters most: the independence of my source of energy in life, being self-motivated and self-generated and self-sufficient in spirit, finding in myself and my highest ideals and my aspirations for excellence a first cause, a fount of energy, a life force.

Dignity, too, is a height that I can approach only gradually and indirectly. For what is dignity but worth, merit, character, excellence? These things grow only in the fulness of time. I must grow into dignity through spiritual maturity and moral ambition and creative aspiration over a span of many years.

39. Depth

The quality of the soul is in its depths, says the *Tao Te Ching*.

The hardest thing in the world is to know and do what I truly want, says *The Fountainhead*.

I cannot be content with the mere surface of life. I must explore the innermost depths, I must go to the heart of things, I must push beyond what is expected and necessary and average to achieve what is rare and great and fine. I must live without reservations and hold nothing back from life.

There is danger in living this way, and it takes great courage to do so. Yet ultimately it is the most practical way to live because only by focusing on what truly matters can I be successful as a human being.

40. Beauty

When I am selective about how I live my life, I achieve a deep simplicity, a kind of honesty and even austerity, a lack of pretension and hypocrisy, an ease and efficiency of choice and action, a timeless sense of calm focus, graceful proportion, and true integrity, a harmony of the parts of my life, a quiet but undeniable energy, a truth to myself, a kind of moral excellence. What is this except beauty?

When I live in this way, I do justice to the real relation, the underlying theme, the ways of true humanity. I am rooted, centered, immersed, engaged, alive to wonder, at ease with life, at home on this earth, aligned with the beauty of existence. I have, too, a sense of lightness, a grace of motion and thought and action, a splendor in my daily routine, a lovely serenity that so easily becomes a smile directed at everything that is.

There is wisdom in selectivity, and beauty in that wisdom.

41. The Fountainhead

Roark is uniquely different because he consistently and fundamentally honors the fountainhead of human progress: the individual person whose self-nature is aligned with the human powers of thought, choice, action, and feeling, and therefore who has the courage to independently understand reality, to focus on what is important, to create great value in the world, and to experience the emotional meaning of life.

There is something primal and eternal about this fountainhead — it is an ever-flowing spring of upward movement, the ultimate source of human ingenuity and happiness, an inexhaustible well of energy, the life-force that has raised humanity in its ever-accelerating ascent, the deepest root of joy and reason and meaning in life. And this is so because the ground of all reality, the depth of all being, lies in individuality. Every thing and every being is individual by virtue of itself and nothing else.

Although this fountainhead is often obstructed and redirected, it can never be fully suppressed because it is latent in every human being as a divine spark of curiosity, a sacred fire of passion, a focused beam of attention, a molten source of energy. These qualities might sound sophisticated, but in fact they are utterly natural: for I am born into the world as a thinking, choosing, acting, feeling being, and when I grow up and out into the world I cannot help but wonder about what I see, turn my attention toward what interests me, care about some people and things more than others, and act to fulfill my needs and desires.

The question is not whether I will think and choose and act and feel, for exercising these powers is as natural as moving and breathing. The question is whether I will honor these innate powers in myself and in others, whether I will live a life of freedom, dignity, depth, and beauty, whether I will align my character with what is best in human nature, whether I will find joy and reason and meaning on this earth, whether I will live up to my highest potential and transform the latent fountainhead within me into a living reality.

42. The Life Force

The depth of all being lies in individuality, and the basis of all life lies in the energy of every living being.

My living energy takes many forms: the mental energy of my thinking, the psychological energy of my attention, the physical energy of my actions, the emotional energy of my feelings, and other, more specific forms of energy within me.

At every moment of my existence, my life is defined by the energy I absorb, the energy I conserve, the energy I transform, and the energy I expend.

There are many external sources of energy in my life: the food I eat and drink, the air I breathe, the sunlight I soak up, the sights and sounds that impinge upon my senses, the things I buy and trade, the books I read, the art works I experience, the inspiration of witnessing the happiness and success of others, the attention I receive from my companions and friends and family. In all these domains of life, I can choose wisely or poorly.

The same is true of the ways I can conserve or waste the energy within me: my posture, my breathing, my sense of balance and proportion, my emotional self-control, my avoidance of negative thoughts and emotions, my honoring the wisdom of my body and my soul, and all the ways that I prepare my mind and body for the activities of life.

That which I take in and conserve is a kind of potential energy that I can transform into the kinetic energy of action: the work I do, the things I create, the people I love and support, and in general how I move through the hours and days and years of my life.

What is my life but this living energy, which I can harness and expend to create joy and reason and meaning in my life? Indeed, just as Newton described physical force as mass times acceleration, I can measure the life force in everything I do as content times intensity: that which I have chosen to do multiplied by the energy, attention, and passion I bring to each activity.

This life force is mine to control, mine to master, mine to direct as I choose, and mine to expend as I think best. No one else has a right to one iota of my living energy, and I have no right to yours, for this life force is the essence of individuality for every living being.

43. Love for Existence

In his interview with Henry Cameron, Roark said: "I love this earth. That's all I love." Think of Roark's Stoddard Temple: slung low over the ground, like the outstretched arms of a great benediction and a silent acceptance of the earth and of all the things on this earth that cannot be changed.

Roark accepted and loved the fact that existence exists. What he accepted and loved above all was the wonderful fact of his own existence. He was simply and deeply glad to be alive.

When I love existence in this way, and especially when I love my own existence, I gain a new kind of fire, a sense that my life is important, a feeling that my happiness is sacred, a dedication to realizing the best of my spirit, a loyalty to my highest potential. I work to live up to the radiant picture of what is possible to me. I accept as the first law of my life an inner demand to seek the best of myself. I come to know that to be happy is my first duty to myself. I seek out the extra quality that makes a dream so vivid, and I try to make that quality real in my waking hours.

When I love and accept this earth, I find that my daily routine can have a kind of honesty and dignity, that the routine necessities of life can acquire a kind of splendor. And why not? Yes, the world can be ugly and small, but life can also be beautiful, and it is supremely good to be alive on this life-giving earth and to feel every day the fresh wonder of an untouched world.

This magnificent appreciation for life is a source of the highest joy possible to human hearts. For what greater joy could there be but to focus on the most important things in my life?

44. The Light Within

Along with Roark, I love not only existence in general and my own existence in particular, but also my consciousness of all that exists. This consciousness takes many forms: an abiding pleasure in the senses, a love of unadorned awareness, a passionate search for knowledge, insight, discernment, and true perception, the wisdom to know the difference between what I can and cannot change.

To be conscious is to watch myself, to monitor myself, to work hard to overcome the things inside me that resist self-knowledge and self-awareness: the ever-present desire to go through life as if asleep, to deceive myself, to live in unconsciousness and darkness. It is to light the lamp of reason within me and to use that lamp to find the source of all reality, which is individuality — not only the singularity of my own existence but also that of every other person, living being, and physical thing. In a beautiful image from the Gnostic Gospels, it is to knock upon myself as upon a door to a wider and brighter truth, to walk upon myself as upon a road to a higher plateau of awareness. It is to have a root and a center and a purpose. It is to know who I am, where I come from, and where I am going. It is to find the place of life and the name of my soul. It is to become my own best teacher and counselor, to accept my own mind as the father of truth, to seek no authority higher than myself and no value higher than my own judgment of what is right.

I cultivate the light within myself so that I can achieve spiritual maturity, personal enlightenment, and transformed awareness. If I do not cultivate the light of reason, I will not shine forth for myself or for those I care about; and if I do not shine forth, I will be in darkness.

45. The Meaning of Life

The meaning of life is not awareness, but action. We live in our minds, and existence is the attempt to bring that life into physical reality, to state it in gesture and form.

As Jesus said in the Gnostic Gospels: "If you bring forth what is within you, what you bring forth will save you. If you do not bring forth what is within you, what you do not bring forth will destroy you."

It is not enough to see and love the depth of all existence, it is not enough to reason about what is great in life; I must also reach for what is high, make the most of what I am and what I have, do my best to live up to what is good, and aspire to constant improvement, to excellence, to whatever share of perfection I can achieve on this earth.

To stand where I am is nothing; but if I have the courage to change what I can and I am able to reach the final expression of my highest possibility, then I can be proud to stand in the place I have reached.

Too many people die a little each day. I must aspire to live more each day. For the higher I aim and the higher I reach, the more my life belongs to me.

Just as Roark's Heller House grew from the cliff on which it was built, as if the cliff had completed itself through the building of the house — so I aspire to grow from what I was and to complete myself through the building of my character and my life. If I do so, I will have a strong foundation for my life, I will have a root and a center and a purpose. And if I achieve this I will not be swayed by others, I will not be buffeted by the winds of circumstance, I will refuse to measure myself against others or as part of anything else. I will live first-hand.

Just as no building must copy another, so no person must copy another. My life is made by own needs. I base my self-respect on personal standards of personal achievement. I seek not to rule and I seek not to teach; I seek only to build and to create whatever value I can bring forth into the world. My highest and best goals

and achievements are not universal and other-directed and social, but personal and self-motivated and not to be touched.

What is the use and the meaning of my life? I am the use and the meaning.

46. The Way

According to the *Tao Te Ching*, the way that can be walked is not the eternal way, and the name that can be named is not the eternal name.

Some take this in a mystical sense, but my interpretation is more practical: the eternal way is general, but the way that I can walk is individual; the eternal name is universal, but the name of my soul is mine alone.

Roark's way, too, was his alone. When I first immersed myself in *The Fountainhead*, I thought that I must live as Roark lived. Yet my path is individual: I cannot necessarily know my calling at the age of ten, as Roark knew he wanted to be an architect; I might by nature be more social than Roark, more collaborative, more lighthearted, more conventional in some respects; I might express my creative powers in different ways, perhaps through mastering a craft or providing an excellent service or nurturing those I care about or building an organization or maintaining what has been created by others. If so, I must know and accept the name of my soul and use those strengths to find my own way in life.

This supreme respect for the individual is something beyond a mere intellectual assent to a philosophy of individualism in the abstract. Instead, it is a pure love for individuality in all its manifest particulars — my own individuality, the individuality of each person I interact with, the individuality of each thing, each place, each building, each project, each task, each performance, each perception, each moment in time.

When I experience individuality at this level, I can live more fully each day because I am always encountering something new and special and uniquely valuable. Each moment that I am blessedly alive, I can feel the fresh wonder of an untouched world and I can come to know more deeply the path that I alone can walk.

47. This White Serenity

In the *Tao Te Ching*, "te" is precisely this individuality: self-nature, raw personhood, character, intention, quality, worth, personal actuality or singularity — good or bad, positive or negative, it is what I am. By contrast, "tao" is human nature, the one path, the great way, a constraining track, an endless course of forward motion, even cosmic unity or potentiality. The tension between individuality and generality, between self-nature and human nature, between actuality and potentiality, between what I am and what I could be, between the way that I can walk and the eternal way, is one of the great themes of life.

In *The Fountainhead*, the most dramatic conflict is not between Roark and Keating or Toohey or Wynand, but between Roark and Dominique. She is like *te* — raw, strange, unconstrained, unbridled, singular, Rand herself in a bad mood. He is like *tao* — self-consistent, unified, integrated, unstoppable, a force of nature. There is a great tension between them, an inexplicable violence that I find unsettling and mystifying unless viewed metaphorically.

Yet at the end of the novel, Roark and Dominique achieve a white serenity that is the sum of all the violence they have known — just as the tension between *te* and *tao*, the struggle between myself and the great way, the sometimes difficult dance between my individuality and the underlying track of right living, is harmonized through experience and reflection, action and understanding, ceaselessly applying the life energy of my being, and patiently waiting for knowledge to settle into hard-won wisdom within me.

What is the alignment between *te* and *tao* but philosophy, the love of wisdom made real in my life? This kind of philosophy is not a dry subject for bookish learning. It is, as Thoreau said, to solve some of the problems of life, not just theoretically, but practically. It is my ongoing relationship with wisdom and insight and right living and the one path to being successful as a human being, to living in a way that is consistent with true humanity. And achieving that alignment brings a great calmness of spirit, the simplicity of being at ease and at home on earth, the ability to feel completely natural, a sense of freedom in serenity, the quiet radiance of certainty, of innocence, of peace with the world and

of peace within myself. This white serenity is knowing and returning to the source, to the fountainhead of joy and reason and meaning in life, to my own individuality.

48. The Tao of Roark

The heart of the earth is made of fire, but sometimes it breaks through and shoots out to freedom. One such spark is Roark's Wynand Building. Another is *The Fountainhead* itself.

In the final scene of *The Fountainhead*, Dominique visits Roark at the construction site for the Wynand Building. Riding the elevator to the top, she passes the pinnacles of bank buildings:

My life is more than finance and economics, more than my career, more than the money I earn. Money is only a means to some personal purpose of my choosing — to invest, to create, to study, to enjoy my limited time on this earth, to live as I see fit. I seek not the power of wealth, but the power of creation. When I let go of money as my primary motivation, I become truly prosperous.

She crests the crowns of courthouses:

My life is more than law and politics, more than my interactions with others, more than my contributions to society. The life of society is secondary, whereas the life of the individual is primary and sacred. If I look to others for fulfillment, I will never be fulfilled. I seek not to rule or to govern, but to create and to build. When I let go of making laws for others, I become more honest, more simple, more direct, more free.

She rises above the spires of churches:

My life is more than religion or philosophy, more than my adherence to a system of ideas, even if that system was created by Ayn Rand herself. I seek not to teach, but to know; and I know that what matters is not to repeat the words of the tao, but to embody it and live it. Because I believe in myself, I no longer need to convince others of what I believe. When I let go of Objectivism as I would to a ladder that has helped me to climb higher and see farther, I become serene.

And then there was only the ocean and the sky and the figure of Howard Roark:

My life is mine to live and enjoy, my individuality is the only untouchable constant of my existence, joy and reason and meaning are not an impossible ideal but a natural way of living that is mine to discover, mine to choose, mine to achieve, and mine to enjoy. My supreme possessions are not outside of me but within me: my integrity, my honor, my freedom, my ideals, my convictions, the honesty of my feelings, the independence of my thoughts, the name of my soul.

That way of living is what I call the Tao of Roark:

It is not the tao of Roark only, but my tao and your tao, the source of all joy and reason and meaning, the fountainhead of individual freedom, personal dignity, spiritual depth, and moral beauty. Although I draw deeply from the well of individuality, it is never used up because it is always full within me. Even when I think I have lost it, I must realize that it can neither be lost nor found, for it is the center and the essence of life, the singing answer to the promise of the music of youth, and a consecration to a joy that justifies the earth

THE END

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"

The Uncertainty Principle
Volume Orange Issue Four "Over the Horizon"

Published by #daLab Press, an imprint of
Mr. Danoff's Teaching Laboratory (<http://mr.danoff.org>).

Editor's Note & Rap, papers and books are all in the public domain.

Everything else Copyright © 2012 The Uncertainty Principle.

The Uncertainty Principle
c/o Mr. Danoff's Teaching Laboratory
Post Office Box 612
Winnetka, Illinois 60093-0612
United States of America

+1.315.750.9903

<http://theup.biz>
editor@theuncertaintyprinciple.biz



*Front and back cover images both taken from The U.P.'s Petite
Soirée № 7 in Albany Park, Chicago on June 30th 2012.*